НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет Інформатики і Обчислювальної Техніки
Кафедра Обчислювальної Техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО

«___»_____2020 р.

Дипломний проект

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютерних систем»
за спеціальністю «Інженерія програмного забезпечення»
на тему: «Система управління особистими записами»

Виконав :
студент IV курсу, групи ІП-62
Павло Андрійович КРАВЕЦЬ _____
Керівник:
Доцент, к.т.н.
Артем Миколайович ВОЛОКИТА _____
Консультант з нормоконтролю:
Професор, д. т. н.
Валерій Павлович СІМОНЕНКО _____
Рецензент:
_____
_____ _____

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ — 2020 року

Національний технічний університет України
“Київський політехнічний інститут”
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки
Освітньо-кваліфікаційний рівень бакалавр
Напрям підготовки 6.050103 « Інженерія програмного забезпечення »

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис) (ініціали, прізвище)
«___» _____ 2020р.

ЗАВДАННЯ

на бакалаврську дипломну роботу студента

КРАВЦЯ Павла Андрійовича

(прізвище, ім’я, по батькові)

1. Тема проекту (роботи) *Система управління особистими записами*
керівник проекту (роботи) *доцент Волокита Артем Миколайович*,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
2. Термін здачі студентом закінченого проекту (роботи) *4 червня 2020р.*.
3. Вихідні дані до проекту (роботи) *технічна документація. теоретичні*
_____
4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)
Розгляд існуючих архітектур, огляд існуючих програмних рішень та технологій, програмна реалізація додатку.
5. Перелік графічного матеріалу (з точним позначенням обов’язкових креслень). структурна схема системи, узагальнена схема роботи системи, блок-схема алгоритму системи.

6. Консультанта проекту (робота), з вказівкою розділів роботи, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання_____

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів дипломного проекту (роботи)	Строк виконання етапів проекту(роботи)	Примітки
1.	*Затвердження теми роботи*	*10.12.2019-15.12.2019*	
2.	*Вивчення та аналіз завдання*	*15.12.2019-15.03.2020*	
3.	*Розробка архітектури та загальної структури системи*	*15.03.2020-15.04.2020*	
4.	*Розробка структур окремих підсистем*	*15.04.2020-01.05.2020*	
5.	*Програмна реалізація системи*	*01.05.2020-15.05.2020*	
6.	*Оформлення пояснювальної записки*	*15.05.2020-25.05.2020*	
7.	*Захист програмного продукту*		
8.	*Передзахист*		
9.	*Захист*		

Студент-дипломник _____ Павло КРАВЕЦЬ
(підпис)

Керівник роботи _____ Артем ВОЛОКИТА
(підпис)

АНОТАЦІЯ

Дипломну роботу виконано на 90 аркушах, вона містить 4 додатки та перелік посилань на використані джерела з 11 найменувань. У роботі наведено 31 рисунок та 7 таблиць.

Метою даної дипломної роботи є створення програмної системи для нотування.

У роботі проведено аналіз існуючих рішень поставленої задачі — різних програмних систем для нотування та порівняно їх можливості. Для розв’язання задачі було вирішено створити клієнт-серверний застосунок з RESTful API сервера, WYSIWYG редактором нотаток та повним шифруванням даних.

Ключові слова: нотування, нотатки, клієнт-серверний, клієнт, сервер, WYSIWYG, шифрування, захист даних

АННОТАЦИЯ

Дипломную работу выполнено на 90 листах, она содержит 4 приложения и перечень ссылок на использованные источники с 11 наименованиями. В работе приведены 31 рисунок и 7 таблиц.

Целью данной работы является создание программной системы для ведения заметок.

В работе проведен анализ существующих решений поставленной задачи – различных программных систем для ведения заметок и проведено сравнение их возможностей. Для решения задачи было решено создать клиент-серверное приложение с RESTful API сервера, WYSIWYG редактором заметок и полным шифрованием данных

Ключевые слова: ведение заметок, заметки, клиент-серверный, клиент, сервер, WYSIWYG, шифрование, защита данных

ABSTRACT

The thesis is performed on 90 sheets, it contains 4 appendices and a list of references to the sources used with 11 titles. The paper contains 31 figures and 7 tables.

The purpose of this thesis is to create a note-taking software system

The analysis of the existing solutions is performed and the functionality of different note-taking software was compared to each other. The client-server architecture with RESTful server API, WYSIWYG text redactor and complete data encryption/

Keywords: note-taking, notes, client-server, client, server, WYSIWYG, encryption, data security.

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЕКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проект	2	
2	А4	ІАЛЦ 467400.000 ВП	Відомість проекту	1	
3	А4	ІАЛЦ 467400.001 ТЗ	Технічне завдання	4	
4	А4	ІАЛЦ 467400.002 ПЗ	Пояснювальна записка	60	
5	А4	ІАЛЦ 467400.003 Д1	Схема структурна	1	
6	А4	ІАЛЦ 467400.004 Д2	Схема функціональна	1	
7	А4	ІАЛЦ 467400.005 Д3	Схема принципова алгоритм перехоплювача запитів	1	
8	А4	ІАЛЦ 467400.006 Д4	Лістинг частин програми	11	

					ІАЛЦ.467200.000 ВП		
Зм.	*Арк.*	*№ докум.*	*Підпис*	*Дата*			
Разробив		*Кравець П.А.*			Система управління особистими записами Відомість дипломного проєкту	*Літ.*	*Аркуш* 6 / *Аркушів* 1
Керівник		*Волокита А. М*					
Реценз.						*НТУУ "КПІ ім. Сикорського" ФІОТ*	
Н. Контр.		*Сімоненко В.П.*					
Затв.							

ТЕХНІЧНЕ ЗАВДАННЯ

до дипломної роботи
освітньо-кваліфікаційного рівня «бакалавр»

на тему: «Система керування особистими записами»

Київ-2020 року

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється на розробку курсу «Веб програмування». Область застосування: практичне використання людьми
в повсякденнному житті для роботи з нотатками.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання роботи кваліфікаційно-освітнього рівня «бакалавр програмної інженерії», затверджене кафедрою обчислювальної техніки Національного технічного
Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3. МЕТА І ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даного проекту є розробка клієнт-серверної системи для нотування

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки є науково-технічна література з теорії і практики програмування, публікації та наукові статті в Інтернеті з даних питань

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробляємого продукту

• Простий і інтуїтивно-зрозумілий інтерфейс системи.

• Незалежність – система повинна мати програмну автономність і не залежати від встановленого програмного забезпечення.

5.2. Вимоги до програмного забезпечення

• Операційна система MS Windows 7 та новіші версії

• Доступ до мережі Інтернет.

• Пошуковий браузер.

5.3. Вимоги до апаратної частини

• Комп’ютер на базі процесора Intel або AMD мінімальної потужності.

• Оперативної пам’яті не менше 1 Гб.

6. ЕТАПИ РОЗРОБКИ

	Дата
Вивчення літератури	10.03.2020
Складання і узгодження технічного завдання	01.04.2020
Створення модулів системи, що розробляється	10.04.2020
Тестування окремих модулів системи	20.04.2020
Допрацювання, налагодження і виправлення помилок	04.05.2020
Оформлення документації дипломної роботи	17.05.2020

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломної роботи

освітньо-кваліфікаційного рівня «бакалавр»

на тему: «Система керування особистими записами»

ЗМІСТ

ВСТУП

Звичка записувати важливу інформацію для особистого використання дуже давня, можливо така ж давня як і сама писемність. Ще давні греки робили нотатки, називаючи їх “гіпомнема”. З плином часу, з’являлися та розповсюджувалися нові форми нотування, наприклад метод Корнела та мапи думок.

Очевидно, що з розповсюдженням персональних комп’ютерів почали з’являтися програми для нотування. До них можна віднести текстові процесори, але спеціалізовані прикладні програми з’являться пізніше.

Сьогодні програми для нотування - це високоспеціалізовані програмні додатки, що можуть надавати багато різних можливостей: створення нотаток з широкими можливістю управління стилями, їхня організація багатьма способами, кросплатформеність та синхронізацію, прикріплення файлів, рукописне введення тощо.

РОЗДІЛ 1
СУЧАСНИЙ СТАН ПРОГРАМ ДЛЯ НОТУВАННЯ

При створенні системи для нотування необхідним є розгляд багатьох різних аспектів: механізмі створення нотаток, можливих типів змісту, їх організації, їх захисту тощо.

1.1. Організація нотаток

Важливою частиною нотування є система організації та пошуку важливої інформації. Основними підходами до організації подібних записів є хронологічний, алфавітний, ієрархічний та заснований на тегах.

Хронологічний підхід передбачає організацію записів за часом їх створення. Цей підхід є дуже простим в реалізації, але він не завжди підходить для конкретної предметної області. Так, сучасні систему для нотування не використовують хронологічний підхід у чистому вигляді, але можуть використовувати його для організації порядку елементі у групі - наприклад, при перегляді нотаток у блокноті, першими будуть новіші/старші.

Алфавітний підхід передбачає організацію записів за їх назвою. Як і хронологічний, він мало підходить для систем для нотування, але може використовуватися в комбінації з іншими підходами.

Ієрархічний підхід передбачає створення багаторівневої організації даних, де кожному об’єкту нижнього рівня відповідає один об’єкт верхнього рівня. Наприклад, нотатки можуть бути організовані в розділи або блокноти. Перевагами такого підходу є його інтуїтивність, легкість адресації та категоризації, а також простоту моделювання при зберіганні. Недоліками є неможливість створення моделювання

таких об’єктів, що належать до двох і більше категорій. Так, у програмах для нотування, що використовують тільки ієрархічний підхід, неможливо створити по тематичному розділу, а потім додати одну нотатку до всіх розділів, до яких вона відноситься. Також до цього підходу відноситься “кольорова” організація, де категоріями виступають певні кольори, що використовуються у графічному відображенні.

У підході, заснованому на тегах, одному об’єкту прикладного рівня (нотатці) відповідає певна множина категорій-тегів. Тег [1] - це ключове слово або термін, що має певне відношення до даних, що він характеризує. За допомогою тегів можлива класифікація та пошук релевантної інформації.

Підхід, заснований на тегах, є більш складним в реалізації, але він дає можливість найточнішої категоризації даних. Однак, так як багато систем дозволяють користувачам визначати нові теги, навігація у системах, що використовують тільки теги, стає дуже складною. Тому теги дуже часто використовують в комбінації з іншими підходами.

1.2. Редактор нотаток

Так як більшість сучасних систем нотування досі використовують текст як основний спосіб зберігання та обробки даних, текстовий редактор й досі залишається однією з центральних частин програми. Текстовий редактор - це самостійна програма або, як у нашому випадку, програмний компонент, що надає можливість обробки текстових документів та даних. Важливою ознакою текстового редактору є те, у якому вигляді він надає можливість працювати з документом.

Текстовий процесор, що є підтипом текстового редактору, надає широкі можливості зміни макету тексту та перегляду у документів в остаточному вигляді (“для друку”). Ця властивість називається WYSIWYG [2] (What You See Is What You Get, “бачиш те, що отримаєш”) і очевидно, є дуже важливою ознакою для програми для нотування, що спрямована на швидке та зручне введення тексту.

Іншою парадигмою редагування документів є WYSIWYM (What You See Is What You Mean, , “бачиш те, що маєш на увазі”), за якої користувач задає лише структуру документа та його контент, а його оформленням займається сама система або інший блок. При її використанні, користувач використовує мови розмітки, такі як LaTeX, Markdown та HTML. Чималими перевагами є єдність стилю оформлення, лаконічність інтерфейсу (може бути достатньо навіть просто поля введення та кнопки “зберегти”), та пряме і зрозуміле керування структурою інтерфейсу, а недоліками - більшу довжину тексту, що бачить та обробляє користувач, змішування контенту та структури у менш інтуїтивно зрозумілій формі та неможливість бачити остаточний результат під час набору тексту без додаткових маніпуляцій.

Але сучасній людині малого тексту у нотатках. Вона бажає урізноманітнити його схемами та малюнками, додати аудіо та відеозапис, або записати щось від руки, не користуючись клавіатурою. Тому важливим є можливість та форма (наприклад, додати файл у текст у вигляді піктограми, або як спеціальне вкладення) вставки різних додатків.

Деякі програми для нотаток ідуть ще далі: вони надають певний простір (whiteboard, або дошка), по якому можна переміщувати нотатки і мультимедійні додатки. При цьому нерідко ці програми

дають змогу працювати з шарами, що надає широкі можливості - наприклад можна швидко від руки написати коментарі щодо якогось макету.

1.3. Захист даних

Так як за допомогою програми для нотування можуть зберігатися приватні дані, важливою частиною створюваної системи є система захисту даних. Важливими частинами такої системи є автентифікація, авторизація та шифрування даних.

Автентифікація [1] - це процес підтвердження істинності певного твердження, зокрема твердження належності користувачеві певного ідентифікатор (логіна) та можливостей, пов’язаних з ним. Автентифікація може бути заснована на одному чи багатьох факторах, що певним чином ідентифікують користувача - знання певних секретних даних (наприклад, пароля), володіння певним іншим каналом зв’язку для підтвердження особистості (наприклад, телефону, на який приходять одноразові коди доступу) або певних унікальних властивостей користувача (наприклад, біометричних даних)

Авторизація - процес керування правами доступу до певних ресурсів, а також їх перевірки при виконанні дій, що потребують цих прав.

Шифрування [4] - процес оборотного перетворення даних з метою приховання інформації. Розрізняють симетричне та асиметричне шифрування, системи з відкритим ключем та системи з закритим ключем,

При симетричному шифруванні один і той же ключ використовується як для шифрування, так і для розшифрування даних. Алгоритми симетричного шифрування зазвичай мають велику пропускну здатність, але для створення захищеного каналу передачі інформації

необхідна наявність ключа на обох кінцях каналу, що досягається іншими способами, наприклад передачею ключа за допомогою іншого каналу або отриманню спільного секретного ключа за алгоритмом Діффі-Геллмана. Прикладами симетричних алгоритмів є AES, DES та Blowfish.

При асиметричному шифруванні використовується пара ключів: один для шифрування, інший для розшифрування. Один з них стає публічним, або відкритим, ключем, тобто тим, що поширюється для роботи системи, а інший приватним, або закритим (зберігається у власника ключів). Який саме стає приватним, а який публічним, залежить від задачі. Асиметричні алгоритми використовуються у системах обміну даними, системах цифрового підпису, тощо. Хоча вони мають меншу пропускну здатність та потребують довшого ключа для еквівалентної криптостійкості, ніж симетричні, але вони не потребують поширення секретного ключа, що спрощує організацію захищених каналів зв'язку. До таких алгоритмів належить RSA, DSA, Elgamal та інші.

Хоча системи з відкритим та системи з закритим ключем дуже часто плутають з асиметричним та симетричним шифруванням, ці визначення не еквівалентні. Так, системи з симетричним шифруванням можуть бути системами з відкритим ключем: за допомогою протоколу Діффі-Геллмана сторони можуть обмінюватися публічними ключами для створення єдиного секрету - ключа для симетричного шифрування.

У свою чергу, алгоритми симетричного шифрування поділяються на потокові і блочні. Блочні алгоритми шифрування потребують розбиття даних, що шифруються на блоки однакового розміру, і шифрують кожний блок одним і тим же ключем. Якщо розмір вхідних

даних не кратний розміру блоку, то до його можуть доповнювати до необхідної довжини.

Потокові алгоритми не потребують такого розбиття. Вони шифрують дані як потік біт за допомогою потоку-ключа (зазвичай, за допомогою операції XOR). Потік-ключ зазвичай формується як псевдовипадкова послідовність з певного сім'я (seed)

У сучасних системах автентифікації паролі не шифрують, а хешують. Хешування - це процес перетворення даних на дані фіксованого розміру (хеш). На відміну від шифрування, хешування може бути необоротним. За цієї умови, а також стійкості до колізій першого роду (складність знаходження такої інформації, хеш якої дорівнює хешу шуканої інформації) та другого (складність знаходження пари різних фрагментів інформації, хеш яких дорівнює один одному), хеш-функцію називають криптостійкою і використовують у системах захисту інформації.

Але так як два різних користувачі можуть мати однаковий пароль, важливим є створення таких умов, що їх хеші будуть різними. Для цього використовується додатковий рядок даних, що називається сіль. Цей рядок має бути випадково згенерованим і індивідуальним для кожного паролю. Використання солі дозволяє не тільки локалізувати можливі порушення режиму доступу (знаючи пароль одного користувача, зловмисник не знатиме користувачів, що мають тій же пароль), але й запобігатиме атаці за допомогою райдужних таблиць - завчасно прорахованих пар рядок-хеш. Сіль не є секретом, її можна зберігати у відкритому вигляді разом з записом о користувачі.

1.4. Перегляд нинішніх програм для нотування

У цьому розділі представлені програми для нотування, описані можливості, які вони надають, їх переваги та недоліки.

1.4.1. Microsoft OneNote

Microsoft OneNote [5] - це пропрієтарна безкоштовна програма для створення та організації нотаток, створена компанією Microsoft. Має преміум-версію, що доступна власникам Office 365.

OneNote - кросплатформений застосунок. Ним можна користуватися через веб-інтерфейс, десктопні програми для Microsoft Windows та macOS та додаток для мобільних пристроїв під управлінням Android, iOS та Windows Phone. OneNote забезпечує синхронізацію, що дозволяє легко переходити з одныэъ платформи

Для нотаток OneNote використовує триступеневу організацію: нотатки належать до сторінок, а сторінки до розділів. При цьому сторінки працюють як дошки: всі нотатки на ній видно одночасно та їх можна вільно переміщувати по всій сторінці

Крім того, OneNote дозволяє зробити будь-який рядок на сторінці сторінки тегом - як рядок у запису так і назву усієї сторінки. OneNote надає велику кількість типів тегів для використання, але визначення нових користувацьких типів можливе не на всіх платформах: наприклад, це можливо у десктопний версії для Windows 10, але не можливо у веб-версії.

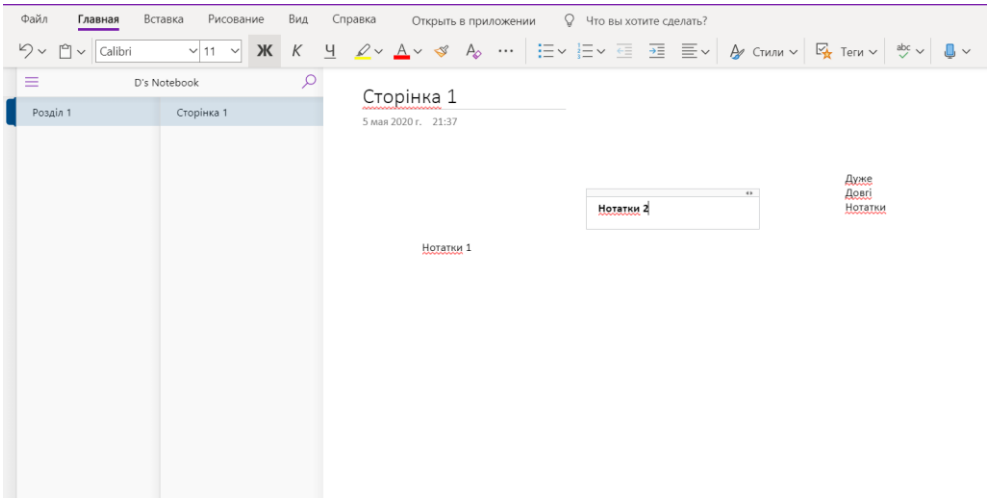


Рис. 1.1 Організація нотаток OneNote

OneNote дозволяє проводити пошук по тегам. Так як до одного рядка можна додати декілька типів тегів, то пошук проводиться як по типу, так і по тексту тегу

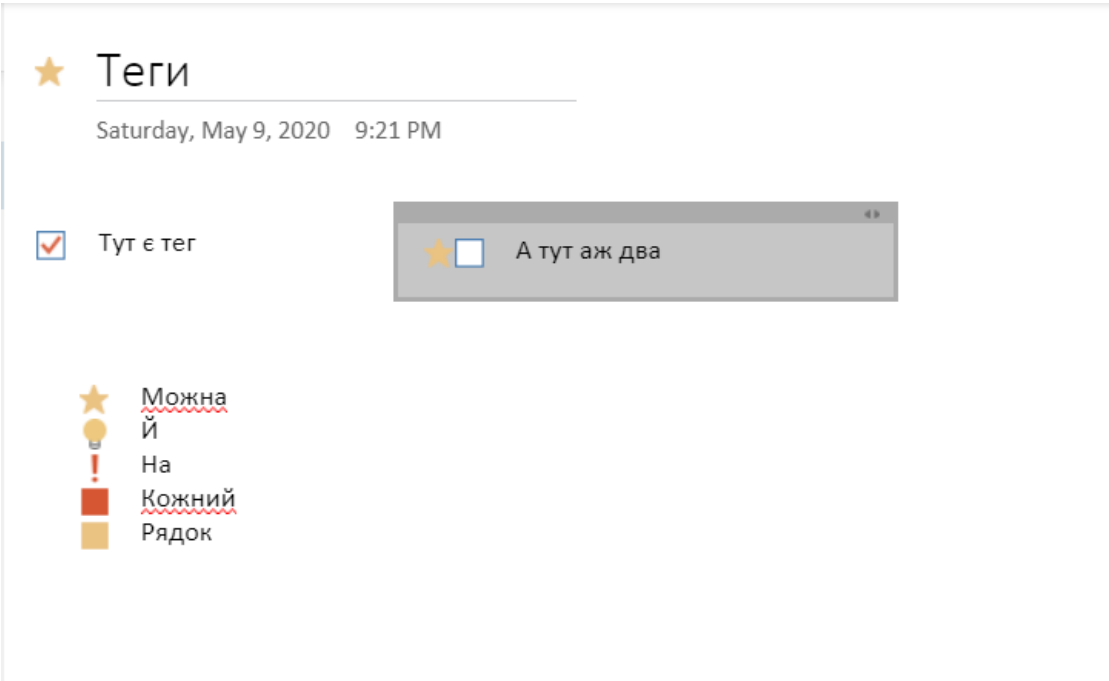


Рис. 1.2 Теги OneNote

OneNote дозволяє додати багато видів вкладень до нотаток: файли та, зокрема, фото, записи аудіо та відео. Крім того, підтримується рукописне введення. Так як програма дозволяє працювати зі шарами,

то це дозволяє легко додати коментарі до фото при використані графічного планшета.

Нотатками у OneNote можна ділитися, даючи можливість співпраці. Це реалізовано за допомогою поширення посилань на сторінки нотаток та можливістю конфігурації прав доступу

OneNote має досить сильну систему захисту інформації: окрім стандартних заходів захисту інформації: шифрування під час транспортування, хешування паролів для зберігання та захисту серверів від зловмисного втручання, OneNote пропонує можливість створення секцій, захищених паролем. Для шифрування даних у цих секціях використовується алгоритм AES-128. Пароль не зберігається ніде у системі OneNote, що мінімізує ризик зловмисного доступу зі сторони осіб, що отримали доступ до даних.

1.4.2. Evernote

Evernote [6] - це пропрієтарна умовно безкоштовна програма для створення та організації нотаток, створена Evernote Corporation. Має декілька рівнів аккаунтів (перший - безкоштовний), що відрізняються рівнем обмежень на трафік то розмір нотаток, крім того, тільки платним аккаунтам доступні певні функції..

Як і OneNote, Evernote - кросплатформений. Існують клієнти для Microsoft Windows, macOS, Android, iOS, Windows Phone та Blackberry OS. Evernote забезпечую синхронізацію між платформами, але для безкоштовних аккаунтів вона обмежена двома пристроями.

На відміну від OneNote, двоступеневу організацію з необов’язковим третім ступенем: нотатки належать до блокнотів, блокноти можуть

належати до наборів. Нотатки - звичайні текстові документи з можливістю вставки мультимедіа та управління стилями.

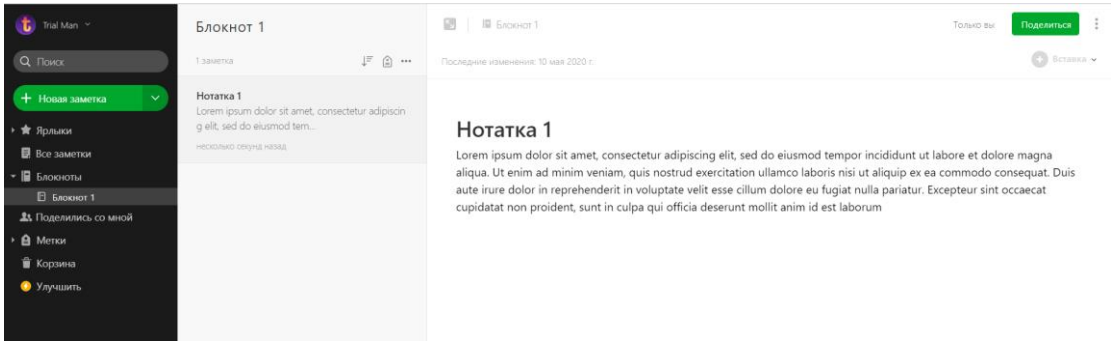


Рис. 1.3 Організація нотаток Evernote

Система тегів Evernote відрізняється від тієї, що є у OneNote. До кожної нотатки можна додати список користувацьких тегів. Для цього лише ввести текст тегу у спеціальне поле унизу сторінки. Evernote дозволяє легко фільтрувати та шукати нотатки по тегам.

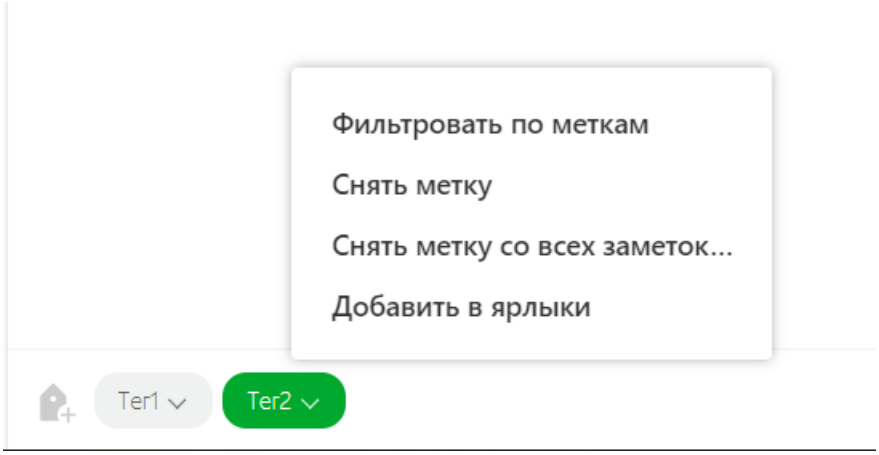


Рис. 1.4 Теги Evernote

Як і OneNote, Evernote дозволяє додати багато видів вкладень до нотаток, але не всі доступні на всіх платформах. Так, запис відео та аудіо засобами без явного використання інших додатків, а також рукописне введення недоступні у веб-версії.

Evernote надає можливість кооперації в роботі з нотатками через поширення запрошень з різними правами доступу. Платним аккаунтам доступний розширений набір функцій.

Як і OneNote, Evernote надає можливість як базового, так і розширеного захисту інформації. На відміну від OneNote, Evernote для зберігання даних Evernote використовує Google Cloud Platform для зберігання інформації, використовуючи алгоритм AES-256 для шифрування даних. Крім того, користувачі Evernote для Windows та Mac мають можливість зашифрувати частину нотатки власним паролем з використанням алгоритму AES-128.

1.4.3. Google Keep

Google Keep - це пропрієтарна безкоштовна програма для створення нотаток, створена компанією Google.

Google Keep - кросплатформений сервіс, але список платформ, що підтримуються, значно менший, ніж у OneNote та Evernote: існує лише веб-сервіс (з розширеною інтеграцією та функціоналом при використанні Google Chrome) та додаток для ОС Android.

На відміну від OneNote та Evernote, Google Keep не використовує ані обов’язкову ієрархічну організацію нотаток: типово, всі нотатки створюються в єдиному просторі, але є можливим створення ярликів, що є схожими на розділи та теги в інших системах: по ним можлива фільтрація, всередині ярлика можна створити нотатку (хоча вона все одно з’явиться у загальному просторі), одна нотатка може належати до декількох ярликів. Крім того, Google Keep надає можливість організації за кольором нотатки

Google Keep дозволяє додати фото до нотатки та або ввести щось від руки. Крім того, версія для Android дозволяє записувати аудіо та

транскрибувати його в текст зо допомогою інших сервісів Google, а також розпізнавати текст на фото. Крім того, завдяки інтеграції з іншими сервісами Google, Google Keep дозволяє створити встановити дату та час, коли система має нагадати користувачу про цю нотатку.

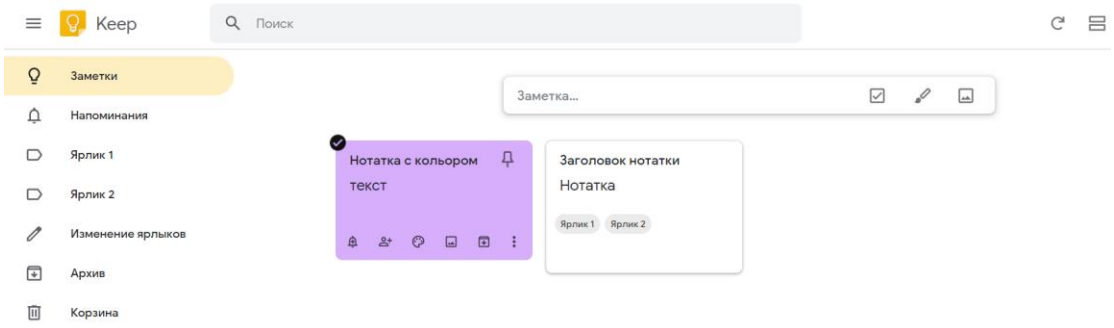


Рис. 1.5 Організація нотаток Google Keep

Google Keep має інструменти для співпраці: користувачі можуть надати доступ до своїх нотаток іншим користувачам Google Keep, але організаційні деталі - ярлики та колір - поширені не будуть.

На відміну від OneNote та Evernote, Google Keep не надає можливостей захисту інформації понад стандартних шифрування під час транспортування і захисту аккаунтів користувачів.

1.4.4. Turtl

Turtl [7] - це пропрієтарна умовно безкоштовна програма з відкритим кодом для створення і організації нотаток. Як і Evernote, має декілька рівнів аккаунтів (перший - безкоштовний), що відрізняються об’ємом даних, що можуть зберігатися, рівнем технічної підтримки та можливістю співпраці.

Turtl - кросплатформена система. Вона доступна у ОС Windows, macOS, Linux та Android.

Для організації нотаток Turtl використовує концепції просторів та дошок (spaces and boards, ієрархічні структури), а також тегів. Кожна нотатка обов'язково належить до певного простору і може належати до певної дошки. Дошки схожі на ярлики в Google Keep: вони також надають можливість переглянути велику кількість нотаток одночасно, але на відміну від OneNote та Google Keep, Turtl не надає можливості якимось шляхом міняти розміщення нотаток.

Теги у Turtl схожі на теги Evernote - це рядок тексту, що визначається користувачем, за яким потім можна вести пошук.

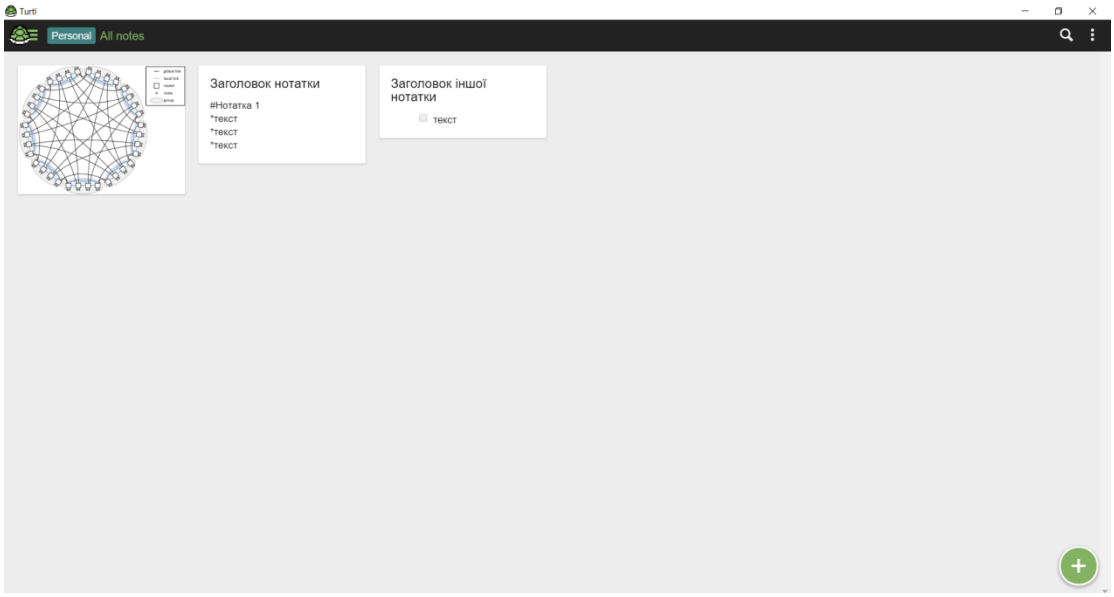


Рис. 1.6 Простір Turtl з нотатками

Великою відмінністю Turtl від інших розглянутих програм для нотування є те, що її редактор працює за парадигмою WYSIWYM і використовує мову розмітки Markdown. Ця мова розмітки є дуже простою і дозволяє швидко додавати просте форматування, таке як жирний шрифт, заголовки та списки, а також вільно додавати HTML-елементи.

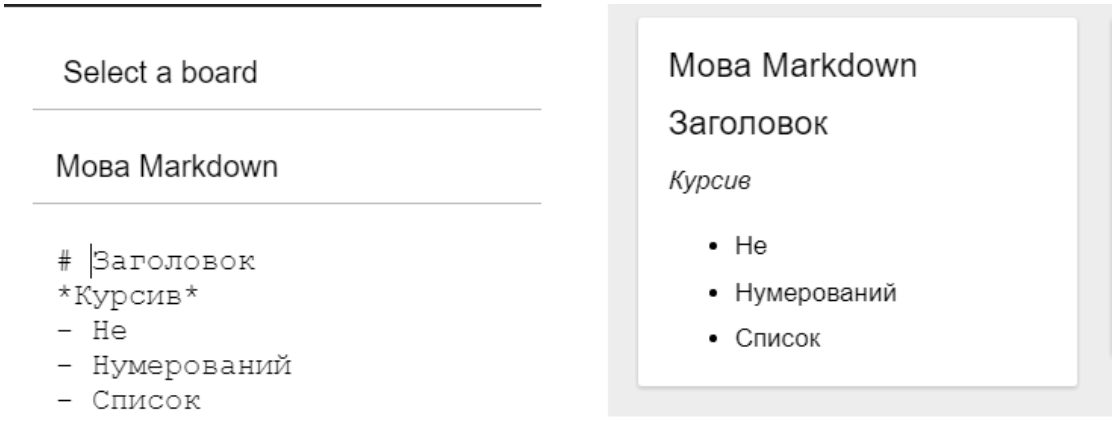


Рис. 1.7 Мова Markdown та її відображення

Крім того, Turtl надає можливість додавати файли до нотаток, а малюнки можна додавати як з файлів на пристрої, так і з URL.

Turtl надає можливість співпраці шляхом надання доступу іншим користувачам до певного простору. Кількість таких користувачів обмежена рівнем аккаунту.

Однією з найважливіших відмінностей Turtl від інших програм для нотування є те, що вона шифрує абсолютно усі дані у нотатках за допомогою симетричного шифру ChaCha20. Кожний об’єкт шифрується своїм ключем, який можн отримати за допомогою основного ключа аккаунту, що отримується на основі пароля та адреси електронної пошти (використовується як сіль). При цьому основний ключ не зберігається на сервері - тобто усі дані захищаються навіть від інсайдерських атак. Важливим мінусом є те, що Turtl не має системи відновлення та зміни паролю - якщо ви втратили пароль, єдине, що можна зробити, це видалити аккаунт.

Таблиця 1.1 - Порівняння можливостей систем для нотування

Властивість	Microsoft OneNote	Evernote	Google Keep	Turtl
Платформи	Веб-інтерфейс, Windows, macOS, Android, iOS, Windows Phone	Веб-інтерфейс, Windows, macOS, Android, iOS, Windows Phone, Blackberry OS	Веб-інтерфейс, Android	Windows, macOS, Android
Система організації нотаток	Ієрархічна, теги	Ієрархічна, теги	Ярлики, кольори	Ієрархічна, теги
Парадигма редактору	WYSIWYG	WYSIWYG	WYSIWYG	WYSIWYM
Додатки до нотаток	Файли, фото, відео, аудіо, рукописний текст	Файли, фото, відео, аудіо, рукописний текст	Фото, рукописний текст, нагадування	Файли, фото
Захист даних при зберіганні	Секції з паролем	Частини нотаток з паролем	Нема	Повне шифрування даних

Як можна бачити з таблиці, на даний момент на ринку немає програми для нотування, що одночасно надає можливість редагування текстових документів в режимі WYSIWYG та безпеку даних, що забезпечується їх повним шифруванням. Саме тому програма для нотування, що створюється, матиме саме таку комбінацію функцій. Крім того, зважаючи на важливість доступу до системи з великої кількості можливих платформ та великі потреби часу на реалізацію великої кількості окремих клієнтів, вирішено зосередитися на розробці веб-клієнта. У якості системи організації нотаток було обрані ієрархічна та тегова системи, подібні до систем, що використовуються у Evernote та Turtl, так як вони забезпечують велику гнучкість та є знайомою користувачам багатьох сучасних систем. Було визначено, що можливими додатками будуть файли та малюнки.

ВИСНОВКИ ДО РОЗДІЛУ 1

У цьому розділі були проаналізовані теоретичні дані, що пов’язані з процесом створення системи для нотування, та деякі програми, що існують наразі на ринку.

Були визначені можливі парадигми редагування тексту, організації записів та методи захисту даних, розглянуті різновиди алгоритмів шифрування.

Були проаналізовані деякі наявні програми для нотування: Microsoft OneNote, Evernote, Google Keep та Turtl. Були визначені переваги та недоліки цих програм, їх підходи до створення нотаток, їх організації, захисту при зберіганні, а також можливості співпраці. Було визначено, що на даний момент немає програми для нотування, що поєднує безпеку повного шифрування даних та зручність редагування даних у режимі WYSIWYG. Тому, для створюваної системи був обраний такий набір функцій:

1. Створення нотаток у режимі WYSIWYG
2. Можливість додавання малюнків та файлів до нотаток
3. Можливість їх організації за допомогою ієрархії та тегів
4. Повне шифрування користувацьких даних
5. Доступ до систем через веб-інтерфейс

РОЗДІЛ 2
ОГЛЯД СУЧАСНИХ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ СИСТЕМ ДЛЯ НОТУВАННЯ

Зважаючи на особливості завдання та навички роботи з певними технологіями, було вирішено розробити систему з клієнт-серверною архітектурою, з клієнтом, побудованим за допомогою фреймворка Angular на мові Typescript, сервером, побудованим за допомогою фреймворка Spring на мові Java та з використанням бази даних під керуванням PostgreSQL для зберігання даних.

2.1. Огляд сучасних веб-технологій

З розвитком глобальної мережі з’являлися нові виклики. І одним з перших таких викликів для World Wide Web стала необхідність стандартизації запитів. Як для вирішення цієї проблеми був створений протокол HTTP.

HTTP (Hypertext transfer protocol) - це протокол прикладного рівня призначений для передачі даних (спочатку - для передачі гіпертексту, нині - даних у довільному форматі). Передача даних організується за моделлю “клієнт-сервер”: клієнт формує запит, відправляє його до сервера, сервер виконує необхідні діє, формує та відправляє відповідь. Протокол HTTP не надає ніякого захисту повідомлень - вони можуть бути вільно змінені та підроблені зловмисником. Для запобігання протокол був створений **HTTPS** (HTTP Secure) - розширення протоколу HTTP, за використання якого дані шифруються за протоколом TLS (у старіших версіях - SSL). Повідомлення шифруються синхронним алгоритмом, ключ до якого є унікальним для кожної сесії та отримується до початку обміну під час так званого

рукостискання (handshake), яке захищено асиметричним шифруванням.

Але окрім механізму транспортування та захисту даних, необхідними є також інструменти, що нададуть можливості керувати представленням даних. У сучасному стеку веб-технологій зазвичай цю роль займає трійка HTML, CSS та JavaScript (або його «мови-нащадки»).

HTML (HyperText Markup Language) - це стандартизована мова розмітки гіпертекстових документів. Такі документи містять інформацію про структуру представлення даних та самі дані (або їх джерело). Веб-браузери отримують HTML-документи від серверу та інтерпретують їх, формуючи графічне відображення веб-сторінки. Документи можуть містити дані як про елементи структури, так і про властивості їх відображення, сукупність яких називається стилем. Але зазвичай опис стилю виносить в окремі файли **CSS** (Cascading Style Sheets) - каскадні таблиці стилів.

Але HTML та CSS не надають зручних можливостей створення динамічних веб-сторінок. Тому зазвичай для цього використовують певну мову програмування (наприклад, JavaScript, TypeScript, PHP), а іноді - певний фреймворк для цієї мови, що спрощує створення сучасних веб-застосунків. Одним з таких фреймворків є Angular.

Angular - це front-end фреймворк з відкритим кодом, що розробляється компанією Google, що написаний на мові TypeScript. Перший варіант цієї платформи був написаний на JavaScript, а після переробки отримав назву AngularJS. Angular надає можливості динамічної генерації веб-сторінок, створення компонент, придатних для повторного використання у багатьох частинах системи,

двостороннього зв'язування даних у представленні і моделі, зміни даних на веб-сторінці без явного перезавантаження сторінки та багато іншого. Але все ж таки Angular - це фреймворк-основа, він не надає всіх можливостей, що можуть бути необхідними при розробці системи. Наприклад, він не надає готового RichText-редактора. Для цього ми можемо використати CKEditor.

CKEditor [8], у минулому FCKeditor - це RichText-редактор з відкритим кодом, що може легко інтегруватися з великою кількістю різних фреймворків, зокрема з Angular. CKEditor був створений Frederico Caldeira Knabben, а нині розробляється великою спільнотою open-source розробників та компанією CKSource. Перевагами CKEditor є малий розмір та велика швидкість роботи без необхідності встановлення на пристрої клієнта, можливість вільного керування набором функцій редактора, наявність інтеграції з багатьма фреймворками та велика та активна спільнота розробників.

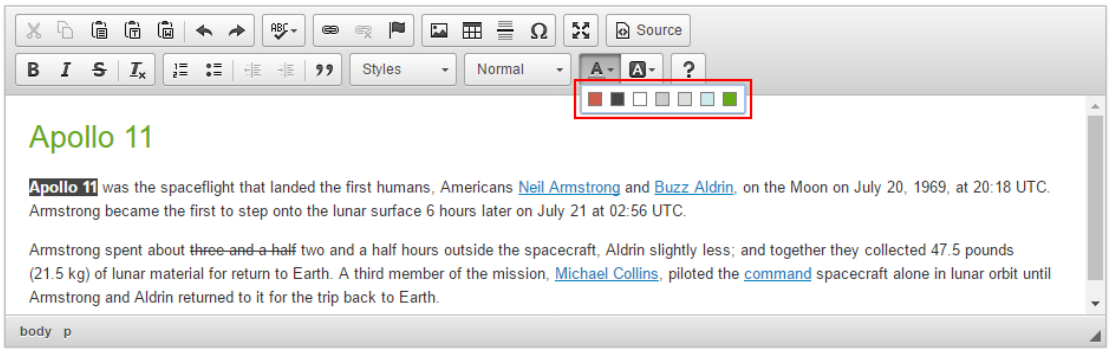


Рис. 2.1 Приклад редактору CKEditor

CKEditor є модульним редактором – більша частина його функцій надається у вигляді плагінів. Розробник має можливість або обрати збірку з певним набором плагінів, або створити власну, вільно обираючи необхідні функції.

Так як однією з основних властивостей програми, що розробляється, є повне шифрування даних на клієнті, необхідно обрати як шифр, так

і джерело, що надаватиме його реалізацію. Так як ми необхідно шифрувати порівняно велику кількість даних без необхідності поширення ключів, то очевидно, що потрібний симетричний алгоритм шифрування. Був обраний алгоритм AES-128.

AES-128 (Advanced Encryption Standard)**,** також відомий як Rijndael - це симетричний блочний алгоритм шифрування, що був прийнятий як стандарт шифрування у США у 2002 році. Він має високу криптостійкість та швидкість, остання навіть вищу при використанні нових процесорів від Intel та AMD, що мають апаратне прискорення цього шифрування. AES й досі залишається одним з найпоширеніших алгоритмів шифрування.

Для отримання ключа шифрування була обрана хеш-функція **PBKDF2** з використанням паролю в якості основи та адреси електронної пошти в якості солі. Ця функція була спеціально розроблена для отримання ключів для шифрування з паролів, і є криптостійкою. Вона дозволяє отримати отримати ключ довільної довжини, що сприяє його використанню разом з AES-128, порівняно з SHA-1 та іншими, що надають ключ фіксованої довжини.

При виборі реалізації криптографічних алгоритмів на стороні клієнта розробник має можливість або вибрати певну бібліотеку, що їх реалізує (наприклад, CryptoJS), або використати інтерфейси **Web Cryptography API** [9]**.** Web Cryptography API – це набір інтерфейсів-рекомендацій щодо реалізації криптографічних алгоритмів на низькому рівні. Ці інтерфейси реалізуються кожною компанією-розробником браузеру самостійно.

Як сторонні бібліотеки, так і Web Cryptography API мають як переваги, так і недоліки. Бібліотеки зазвичай надають більш зручний інтерфейс

високого рівня та (так як реалізуються на чистому JavaScript) однаково працюють у всіх браузерах, але вони зазвичай набагато повільніші та, через меншу професіональність розробників, можуть мати критичні помилки у реалізації алгоритмів, що робить їх менш криптостійкими. З іншого боку реалізації Web Cryptography API є швидшими, але існують не у всіх браузерах. Але з плином часу все більше користувачів користується браузерами, що підтримують ці стандарти – за даними сайту CanIUse, таких користувачів більше ніж 97%. Отже, було вирішено використовувати саме інтерфейси **Web Cryptography API.**

Під час розробки сучасного веб-застосунку дуже багато часу приділяється дизайну інтерфейсу користувача. Одним з найпоширеніших стандартів сучасного дизайну є **Material Design** (Матеріальний Дизайн) – стиль дизайну, розроблений компанією Google. Він характеризується широким використання тіней та анімації, мінімалістичністю, адаптивністю та плоским інтерфейсом. Для реалізації цього дизайну у проекті була обрана бібліотека **MaterializeCSS.**

2.2. Огляд технологій для розробки серверної частини системи

Іншою важливою частиною клієнт-серверної архітектури є, вочевидь, сервер. Для його розробки був обраний фреймворк Spring для мови Java.

Spring [10]- це універсальний фреймворк з відкритим кодом для Java-платформи (тобто мов, що використовують JVM), що розробляється компанією Pivotal Software. Spring має основу, що надає центральний набір функцій, таких як контексти та Dependency Injection. Крім того, Spring має дуже велику кількість спеціалізованих модулів, що

надають такі можливості, як побудова API, автентифікація та авторизація, робота з базами даних тощо. Раніше, Spring потребував значної конфігурації для використання усіх цих можливостей, але нині Spring Boot знижує кількість необхідної конфігурації до мінімуму. Крім того, він дозволяє використовувати вбудований сервер Tomcat, Jetty або Undertow, позбавляючи від необхідності налаштування окремих серверів. Так як можливості, що надає Spring, дуже сильно залежать від модулів, що використовуються, деякі з них необхідно розглянути окремо.

Для побудови API було вирішено використовувати архітектуру REST, так як задача програми для нотування не потребує обов'язкових сесій. Для спрощення побудови цього API було вирішено використовувати модуль **Spring Web**, що використовує сервлети для побудови API.

Крім того, так як створювана система має надавати можливість зберігання файлів як додатків до нотаток, необхідним є механізм для ефективної їх передачі. Spring Web надає такий механізм завдяки інтерфейсу **MutipartFile**, що дозволяє передавати файл як потік даних

Важливою частиною серверу є обробка помилок, що виникають під час його роботи. Для цього Spring Web надає багато інструментів, одним з яких є **ControllerAdvice**, що реалізує роботу з виключеннями у парадигмі аспектно-орієнтованого програмування, дозволяючи з легкістю обробляти виключення Java

В якості формату передачі повідомлень між клієнтом і сервером був обраний **JSON** (JavaScript Object Notation) - текстовий формат обміну даними, що може читатися людиною. JSON походить з мови JavaScript та є "легшим" за інші формати, які може читати людина,

наприклад XML. Для перетворення JSON-об’єктів на об’єкти Java був обраний **Jackson** - бібліотека з відкритим кодом, що створена спільнотою open-source розробників

Рис. 2.2 Приклад запису об'єкту у форматі JSON

Іншою важливою частиною серверної інфраструктури є захист даних. Модуль **Spring Security** надає широкі можливості у цій області - хешування паролів, автентифікація, авторизація статична та заснована на ролях тощо. В якості алгоритму хешування паролів було обрано **bcrypt** за його криптостійкість та за ефективну реалізацію у модулі Spring Security. Важливою особливістю цієї реалізації є те, що система автоматично генерує та використовує сіль, зберігаючи її разом з паролем, що спрощує використання алгоритму.

Так як пересилання пароля кожного разу, коли ми хочемо скористатися можливостями, що потребують авторизації, є незручним так небезпечним, для авторизації у системі будуть використані токени доступу стандарту **JWT** (JSON Web Token). Ці

токени складаються з трьох частин: заголовку, змісту та підпису. У заголовку записуються дані про медіатип токену та алгоритм хешування підпису, у зміст - певне твердження, яке токен підтверджує (наприклад, наявність прав доступу до певного ресурсу), хто видав токен, у який період часу токен є дійсним. Заголовок і зміст записуються через крапку, кодуються у Base64 і хешуються алгоритмом, що описаний у заголовку, для отримання підпису.

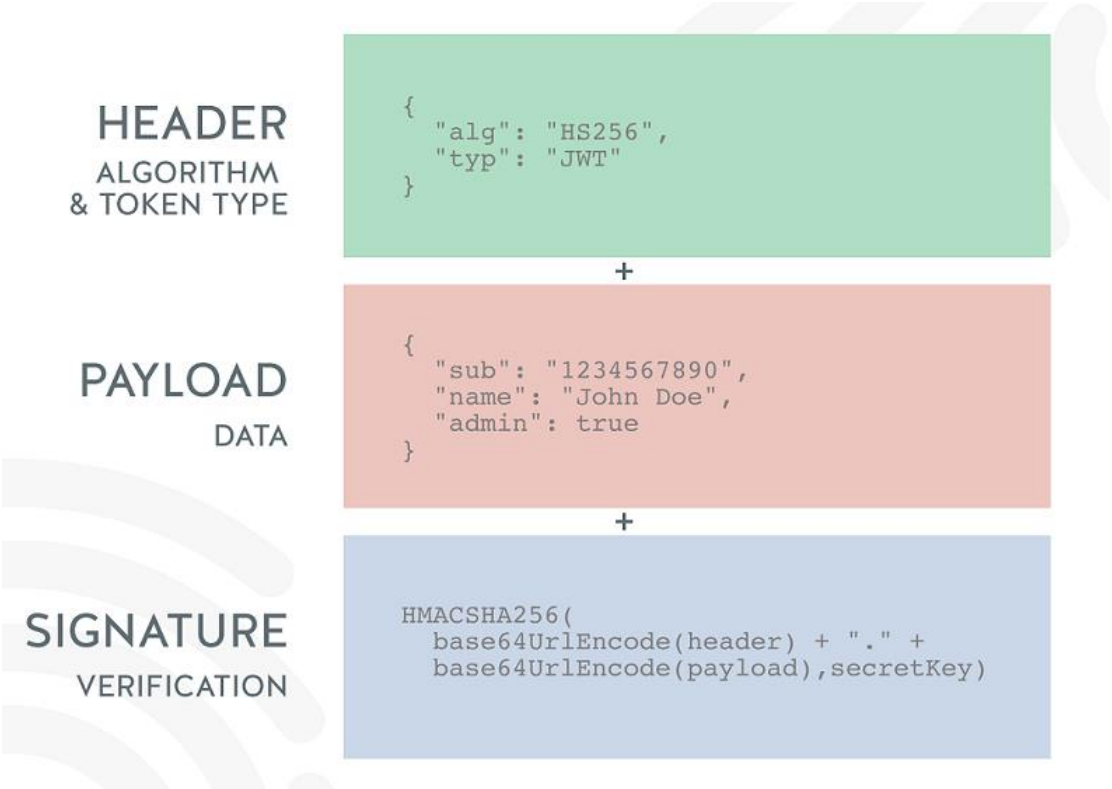


Рис. 2.3 Формат JWT-токенів

Іншою необхідною частиною серверу є робота зі сховищем даних. **Spring Data** надає єдиний високорівневий інтерфейс для роботи з багатьма видами баз даних - як SQL, так і NoSQL. Але Spring Data є “проектом-парасолем”, що описує загальний підхід до роботи зі сховищами даних, а для роботи з окремими видами ми маємо використовувати спеціалізованими модулями. Одним з таких модулів для роботи з SQL-базами даних є **Spring Data JPA.** Цей модуль

підтримує роботи з даними як за допомогою запитів (query), так і за допомогою високорівневих абстракцій - репозиторіїв, що дозволяють декларативну роботу з запитами.

2.3. Огляд бази даних

Так як програма для нотування потребує активної роботи зі сховищем даних, вибір правильного є дуже важливим. Так як розмір нотаток може бути великим, а схема даних та запитів буде стабільною, була обрана окрема SQL база даних під керуванням СУБД **PostgreSQL** [11]**.**

PostgreSQL, також відома як **Postgres** - це безкоштовна реляційна СУБД з відкритим кодом, що розробляється групою розробників, що називає себе PostgreSQL Global Development Group. Postgres базується на принципах можливості розширення та слідування стандарту SQL. Сьогодні PostgreSQL є однією з найпоширеніших баз даних.

Однією з важливих функцій, які потребує база даних для програми для нотування, є можливість ефективного зберігання файлів-додатків та тексту нотаток, що може бути досить довгим. Postgres досягає цього завдяки механізму TOAST. При зберіганні великих об’ємів даних TOAST спочатку спробує зжати великі поля, а потім запише дані в окрему таблицю. Кожна звичайна таблиця має одну TOAST-таблицю, а доступ до них прозорий для користувача. Таким чином, PostgreSQL дозволяє легко зберігати файли в базі даних без зниження швидкості систем, хоча розмір поля все ж таки є обмеженим 1 Гб, але це не є проблемою для системи нотування, де файли виконують допоміжну функцію.

ВИСНОВКИ ДО РОЗДІЛУ 2

У цьому розділі були проаналізований стан технологій, що підходять для розробки системи для нотування. Були обрані наступні технології та сфери їх призначення:

1. Клієнтська частина (фронт-енд): Angular
   1.1. Текстовий редактор: CKEditor
   1.2. Реалізація криптографічних алгоритмів: Web Cryptography API
      1.2.1. Отримання ключів шифрування: PBKDF2
      1.2.2. Шифрування даних: AES-128
2. Серверна частина (бек-енд): Spring
   2.1. API: RESTful API за допомогою Spring Web
   2.2. Захист даних: Spring Security
      2.2.1. Хешування паролів: bcrypt
   2.3. Робота зі сховищем даних: Spring Data JPA
3. База даних: PostgreSQL

Дослідження виявило наявність всіх технологій необхідних для побудови високозахищеної сучасної програми для нотування

РОЗДІЛ 3
РОЗРОБКА СИСТЕМИ ДЛЯ НОТУВАННЯ

3.1. Загальний огляд системи

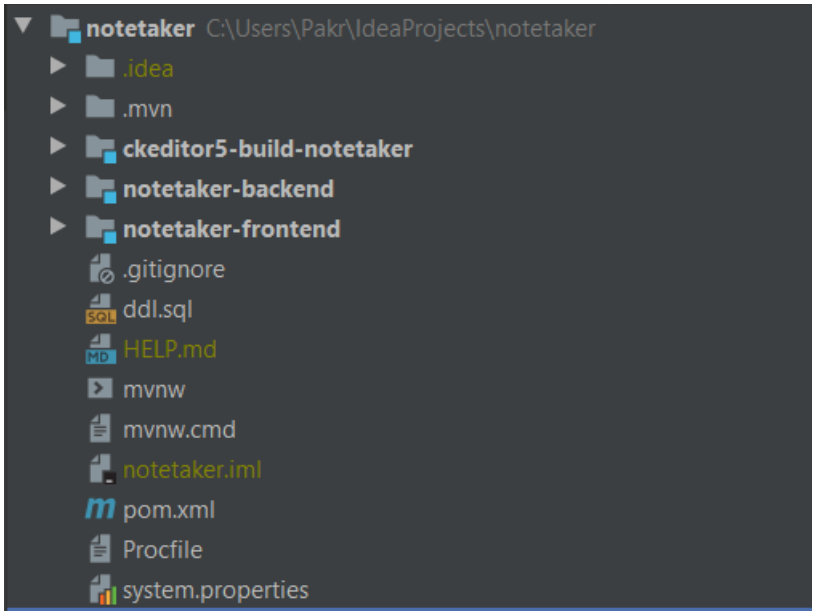


Рис. 3.1 - Загальний архітектура проекту в середовищі розробки

Система для нотування реалізована у вигляді багатомодульного Maven-проекту, придатного для розгортання на сервісі Heroku. Усього у проекті 3 модулі:

1. notetaker-backend – модуль написаний на мові Java, серверна частина системи
2. notetaker-frontend – модуль написаний на мові Typescript, клієнтська частина системи
3. ckeditor5-build-notetaker – модуль написаний на мові JavaScript, спеціальна збірка редактору CKEditor

Клієнтський залежить від редактору, що збирається за допомогою пакетного менеджеру npm, що запускається за допомогою плагіну frontend-maven-plugin. Збірка редактору використовується як

залежність до клієнтського модулю, що також виконується npm, що запускається тим самим плагіном. Отримані ресурси копіюються в папку ресурсів серверного модулю, що дозволяє використовувати сервер TomcatEE для надання сторінок клієнту.

У корені проекту також знаходиться файл Procfile, що надає інструкції для розгортання проекту на сервісі Heroku.

Для керування версіями проекту була обрана система git та сервіс Github.

3.2. Опис моделі даних

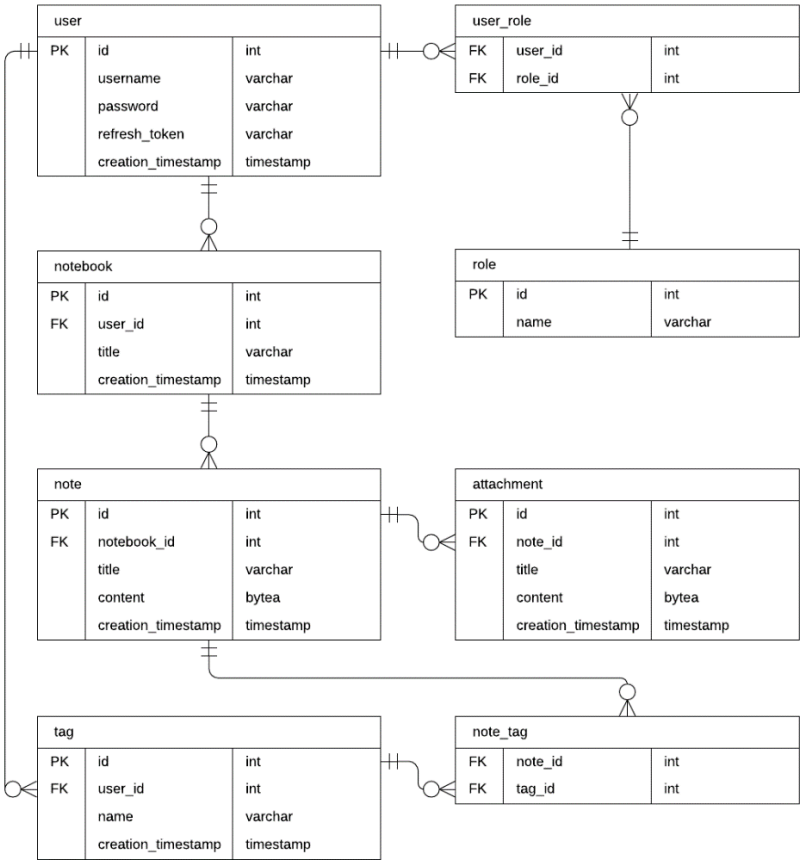


Рис. 3.2 ER-діаграма предметної області

Під час дослідження предметної області системи, що створюється була створена наступна модель даних (доменна модель).

Розглянемо ці сутності детально:

Таблиця 3.1 Сутності предметної області

Назва	Що зберігає	Дані, що шифруються
User	Облікові записи користувачів	Пароль користувача
Role	Можливі ролі (рівні доступу) користувачів	
User_Role	Зв’язок «багато-до-багатьох» між користувачами та їх ролями	
Notebook	Блокноти користувачів	Назва блокнота
Note	Нотатки користувачів	Назва нотатки, зміст нотатки
Attachment	Файли-вкладення до нотаток	Назва файлу, зміст файлу
Tag	Теги, які використовує користувач	Назва тегу
Note_Tag	Зв’язок «багато-до-багатьох» між тегами та нотатками	

Отже, у створюваній системі нотатки матимуть дворівневу ієрархічну організацію (блокноти, нотатки), можуть мати вкладення та теги. Кожний тег належить певному користувачеві. Користувач може мати будь-яку кількість ролей.

3.3. Опис архітектури серверної частини

В якості загальної архітектури серверної частини була обрана монолітна тришарова організація:

1. Шар контролерів – забезпечує зв’язок між сервером та клієнтом. Визначає API серверу. Перетворює об’єкти доменної моделі на об’єкти що передаються на клієнт. Було створено по одному контролеру на сутність, що експортується (Notebook, Note, Attachment та Tag) та контролер для автентифікації та авторизації
2. Шар сервісів – забезпечує виконання бізнес-логіки застосунку. Методи сервісів можуть бути використані контролерами чи іншими сервісами. Було створено по одному сервісу на кожну основну сутність доменної моделі (тобто всі, окрім тих, що існують для зв’язку «багато-до-багатьох») та сервіс авторизації
3. Шар репозиторіїв – забезпечує доступ до даних у базі даних. Кількість репозиторіїв дорівнює кількості основних сутностей доменної моделі

Крім того, був створений аспект ExceptionHandlerAdvice для централізованої обробки виключень та фільтр JWTFilter для реалізації аутентифікації та авторизації за допомогою JWT-токенів.

3.3.1. Опис API серверу

Під час розробки системи для нотування було розроблене наступне API:

Таблиця 3.2 Контролер NotebookController

Шлях	Параметри запиту	Дія
GET /notebook	username – ім'я користувача	Повертає всі блокноти користувача. Повертає HTTP код 404, якщо такого користувача не існує
GET /notebook/{id}	id – ідентифікатор блокноту	Повертає блокнот. Повертає HTTP код 404, якщо такого блокноту не існує
POST /notebook	username – ім'я користувача, Дані про блокнот	Створює новий блокнот для користувача. Повертає HTTP код 404, якщо такого користувача не існує
DELETE /notebook/{id}	id – ідентифікатор блокноту	Видаляє блокнот. Повертає HTTP код 404, якщо такого блокноту не існує

У таблиці 3.2 наведені шляхи запитів до NotebookController, що відповідає за роботу з блокнотами

Таблиця 3.3 Контролер NoteController

Шлях	Параметри запиту	Дія
GET /note/{id}	id – ідентифікатор нотатки	Повертає нотатку. Повертає HTTP код 404, якщо такої нотатки не існує
GET /note	tag_id –	Знаходить всі нотатки, що

Шлях	Параметри запиту	Дія
	ідентифікатори тегів для фільтрування	мають всі надані теги. HTTP код 404, якщо такого тегу не існує
POST /note	notebook_id – ідентифікатор блокноту, дані про нотатку	Створює нову нотатку для у блокноті. Повертає HTTP код 404, якщо такого блокноту не існує
GET /note/{id} /content	id – ідентифікатор нотатки	Повертає зміст нотатки. Повертає HTTP код 404, якщо такої нотатки не існує
PUT /note/{id} /content	id – ідентифікатор нотатки content – зміст нотатки	Замінює зміст нотатки наданим змістом. Повертає HTTP код 404, якщо такої нотатки не існує
PUT /note/{id}/tags	id – ідентифікатор нотатки tag_id – ідентифікатори тегів	Замінює теги нотатки наданими. Повертає HTTP код 404, якщо такої нотатки або якогось з тегів не існує
DELETE /note/{id}	id – ідентифікатор нотатки	Видаляє нотатку. Повертає HTTP код 404, якщо такої нотатки не існує

У таблиці 3.3 наведені шляхи запитів до NoteController, що відповідає за роботу з нотатками

Таблиця 3.4 Контролер AttachmentController

Шлях	Параметри запиту	Дія
GET /attachment /{id}	id – ідентифікатор вкладення	Повертає дані про файл-вкладення (без змісту). Повертає HTTP код 404, якщо такого вкладення не існує
GET /attachment /{id}/content	id – ідентифікатор вкладення	Повертає файл-вкладення. Повертає HTTP код 404, якщо такого вкладення не існує
POST /attachment	note_id – ідентифікатор нотатки, content – файл-вкладення	Створює нове вкладення для нотатки. Повертає HTTP код 404, якщо такої нотатки не існує
DELETE /attachment /{id}	id – ідентифікатор вкладення	Видаляє вкладення. Повертає HTTP код 404, якщо такого вкладення не існує

У таблиці 3.4 наведені шляхи запитів до AttachmentController, що відповідає за роботу з вкладеннями

Таблиця 3.5 Контролер TagController

Шлях	Параметри запиту	Дія
GET /tag/{id}	id – ідентифікатор тегу	Повертає тег. Повертає HTTP код 404, якщо такого тегу не існує
POST /tag	username – ідентифікатор користувача, Дані про тег	Створює новий тег для користувача. Повертає HTTP код 404, якщо такого користувача не існує
DELETE /tag/{id}	id – ідентифікатор тегу	Видаляє тег. Повертає HTTP код 404, якщо такого тегу не існує

У таблиці 3.5 наведені шляхи запитів до TagController, що відповідає за роботу з тегами

Таблиця 3.6 Контролер AuthController

Шлях	Параметри запиту	Дія
POST /signup	Логін та пароль користувача	Створює обліковий запис користувача, повертає access та refresh токени. Повертає HTTP код 409, якщо користувач з таким ім’ям вже існує
POST /signin	Логін та пароль користувача	Автентифікує користувача, повертає access та refresh

Шлях	Параметри запиту	Дія
		токени. Повертає HTTP код 401 якщо користувача з таким логіном та паролем не існує
POST /refresh	Refresh токен	Повертає новий access токен. Повертає HTTP код 401, якщо refresh токен не є валідним
POST /signout		Закінчує сесію користувача, інвалідує refresh токен

У таблиці 3.6 наведені шляхи запитів до TagController, що відповідає за авторизацію та аутентифікацію

3.3.2. Автентифікація та авторизація

Розглянемо детальніше схему автентифікації та авторизації. Для її реалізації використовуються 2 види токенів: access та refresh токени. Обидва види утримують інформацію про користувача, якому вони були надані (логін та ролі, що має користувач), але вони відрізняються терміном придатності: access мають набагато менший, аніж refresh. Терміни придатності токенів імпортуються зі змінних оточення «JWT_ACCESS_VALIDITY» та «JWT_REFRESH_VALIDITY», і отже, легко налаштовуються. На даний момент термін придатності access дорівнює 5 хвилинам, а refresh – 1 добі. Розглянемо схему взаємодії клієнта та сервера.

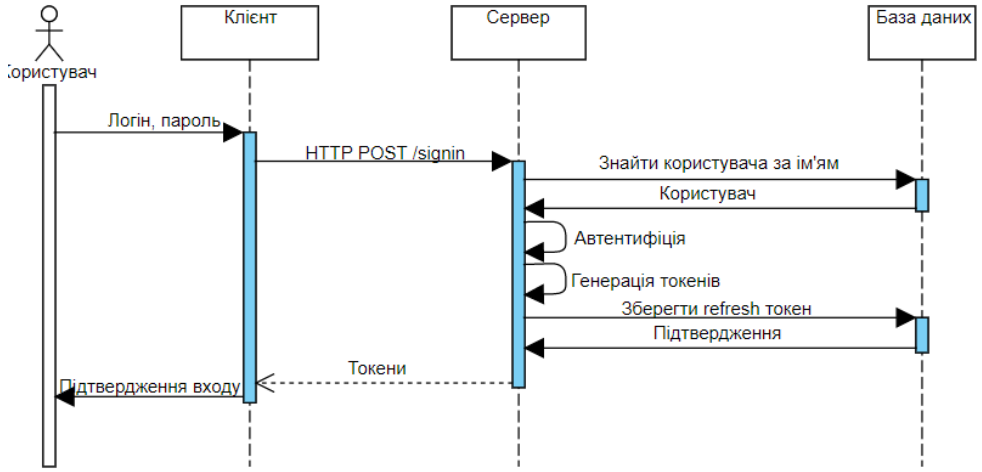


Рис. 3.3 - Схема автентифікації

Клієнт ініціює автентифікацію за допомогою запиту HTTP. За наданим логіном сервер знаходить користувача та автентифікує його за допомогою пароля. Якщо сервер не знаходить такого користувача або пароль є не вірним, повертається HTTP код 401. Якщо все вірно, то генерується пара токенів, і refresh зберігається у базі. Токени повертаються на клієнт та використовуються для доступу до системи.

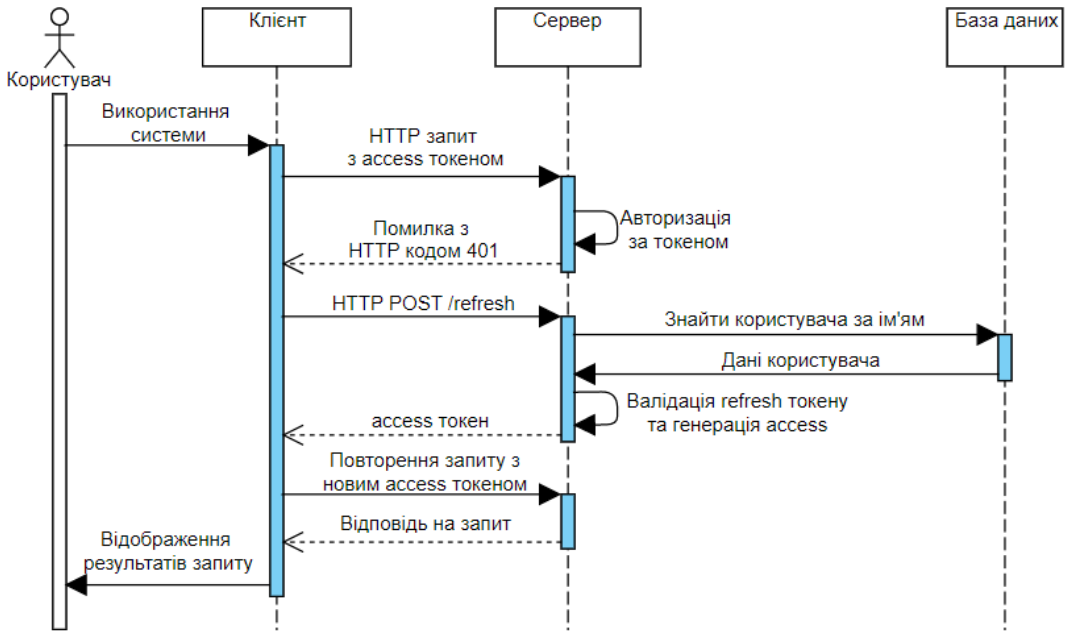


Рис. 3.4 - Схема оновлення access токену

Access токен передається з кожним запитом на сервер у HTTP заголовку Authorization. Коли термін придатності токену закінчується, буде повернене повідомлення о помилці з кодом 401. Клієнт має виконати HTTP запит на оновлення access токену, надавши свій refresh. Цей токен буде провалідований та порівняний з токеном, що зберігається у базі даних. Якщо все правильно, буде повернений новий access токен. Якщо refresh токен невалідний, то клієнт має виконати вхід знову.

Після завершення роботи клієнт має виконати вихід з системи, інвалідуючи refresh токен (видаляючи його з бази даних).

3.4. Опис архітектури клієнтської частини

Клієнтська частина системи реалізована як односторінковий застосунок. Вона складається з декількох Angular-компонентів та сервісів, що використовуються компонентами.

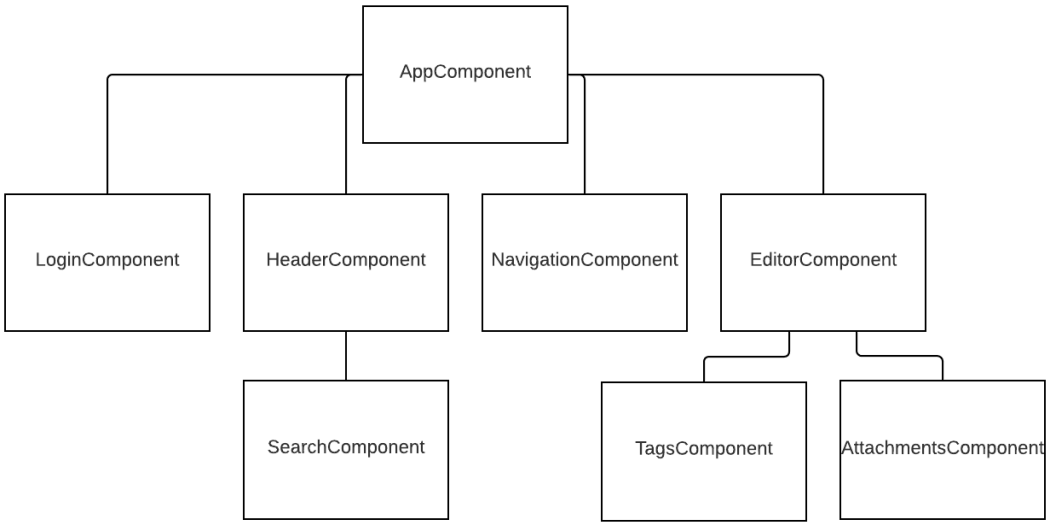


Рис. 3.5 Організація компонентів клієнта

Всі компоненти об’явлені в єдиному модулі AppModule та прямо чи опосередковано використовуються у батьківському компоненті

AppComponent, що відповідає за загальну структуру застосунку та зв’язок інших компонентів один з одним.

3.4.1. Опис компонентів

LoginComponent – це компонент, що відповідає за процедуру логіну у систему. Його представлення реалізовано у вигляді модального вікна з формою для введення логіну та паролю. Він використовує LoginService для роботи з сервером та EncryptionService для створення та зберігання ключа шифрування

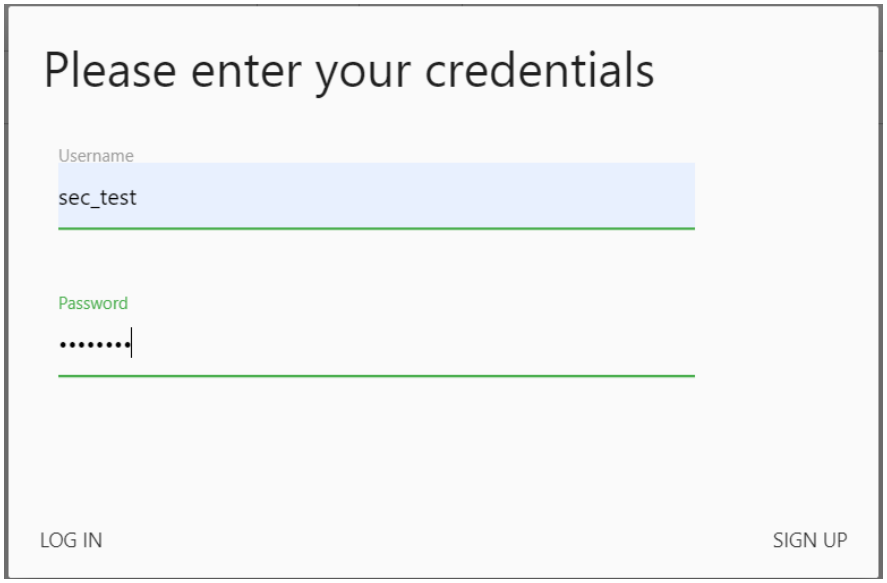


Рис. 3.6 LoginComponent

NavigationComponent – це компонент, що відповідає за навігацію між блокнотами та нотатками. Крім того, він дозволяє створювати та видаляти блокноти та нотатки. Його представлення реалізоване у вигляді бокової панелі зліва, що є фіксованою на широких екранах (більш ніж 992 пікселя) та ховається на малих, що покращує відображення на екранах телефонів. У панелі знаходиться список блокнотів користувача, при натисканні на один з них відкривається список нотаток. Для видалення та створення блокнотів та нотаток реалізовані модальні вікна. Він використовує LoginService для роботи

з сервером, що використовує EncryptionService для шифрування та дешифрування даних

HeaderComponent – це компонент-заголовок. Він відображає назву відкритої нотатки, кнопку «пошук» компонента SearchComponent, кнопку «Logout» та на малих екранах – кнопку відкриття навігації (навігацію ще можна відкрити жестом, «витягнувши» її з лівої сторони екрану). Він використовує LoginService для процедури закінчення сессії

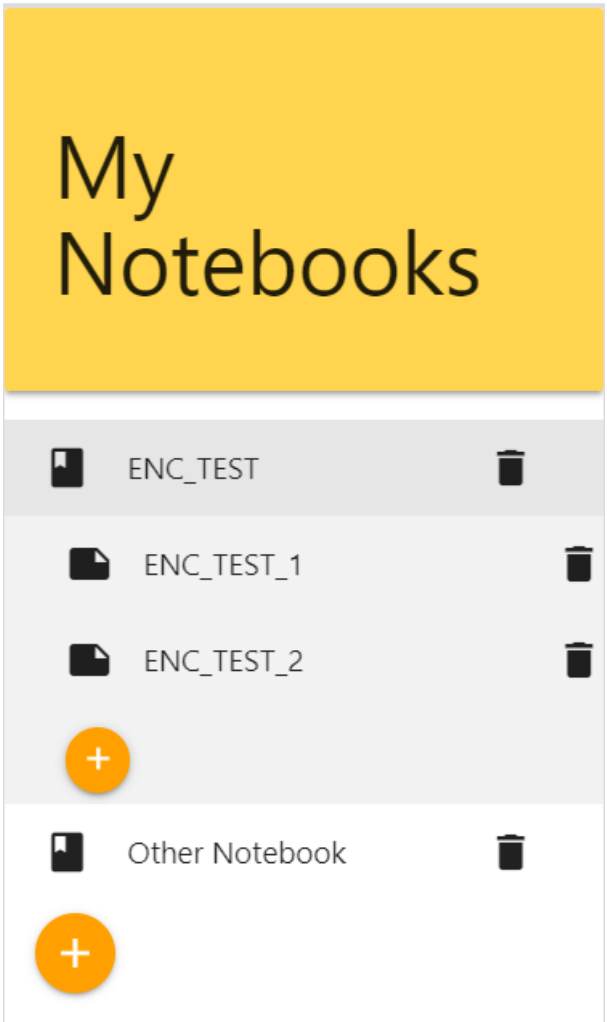


Рис. 3.7 NavigationComponent

Рис. 3.8 – HeaderComponent

SearchComponent – це компонент, що відповідає за пошук нотатки за тегами. Він відображає всі нотатки, що мають всі наведені теги. Він використовує TagService для роботи з сервером, що використовує EncryptionService для шифрування та дешифрування даних

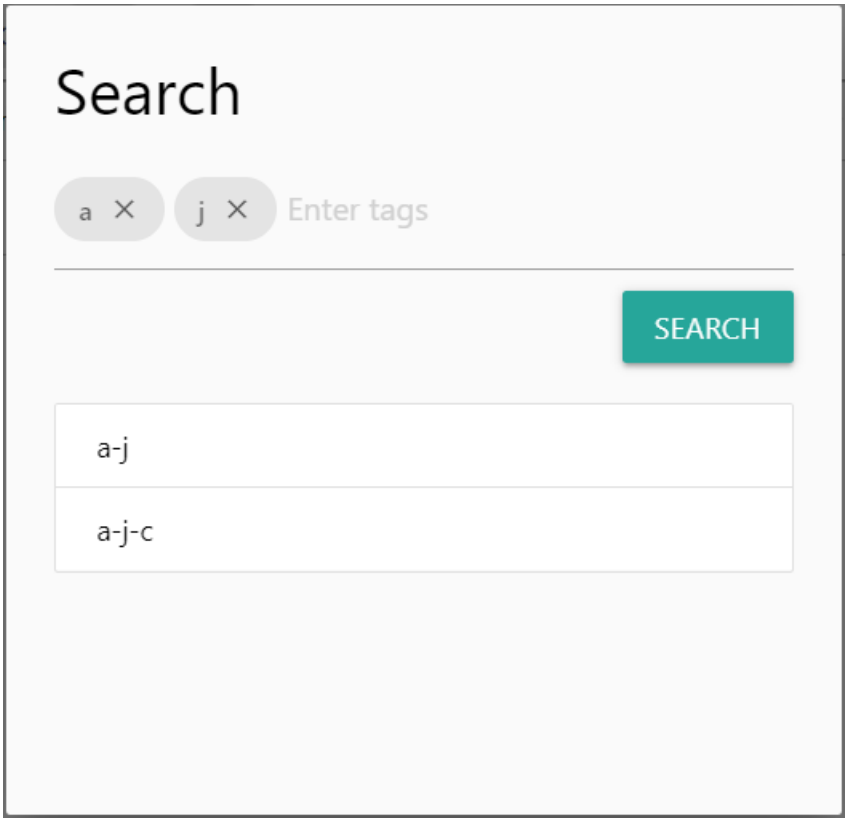


Рис. 3.9 SearchComponent

EditorComponent – це компонент-редактор. Він є основною частиною сторінки клієнта. В ньому знаходиться редактор CKEditor, кнопка «Зберегти» та компоненти TagsComponent та AttachmentsComponent. Він використовує EditorService для роботи з сервером, що використовує EncryptionService для шифрування та дешифрування даних.

TagsComponent – це компонент-стрічка тегів нотатки. Він дозволяє легко додавати та видаляти теги нотатки, створюючи нові теги, якщо шуканого тегу не існує. Він використовує TagService для роботи з сервером, що використовує EncryptionService для шифрування та дешифрування даних

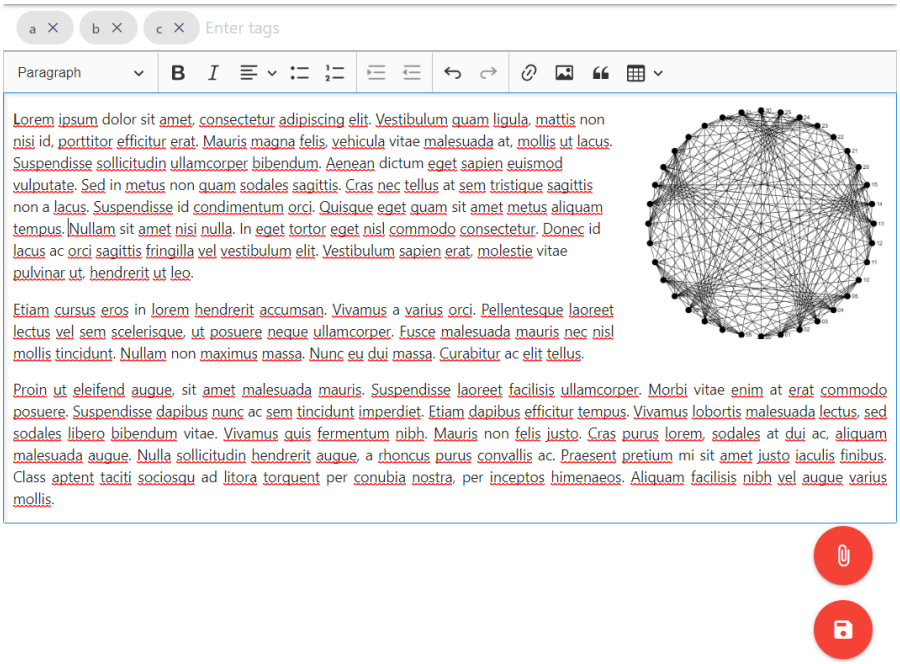


Рис. 3.10 EditorComponent

Рис. 3.11 TagsComponent

AttachmentComponent – це компонент, що відповідає за роботу з файлами-вкладеннями. Він складається з двох частин: кнопки «Вкладення» та модальне вікно, яке вона відкриває. У вікні наданий список вкладень до поточної нотатки з можливістю обрання нового файла для додавання, видалення вкладень та їх завантаження на комп’ютер. Він використовує AttachmentService для роботи з сервером, що використовує EncryptionService для шифрування та дешифрування даних.

Рис. 3.12 AttachmentComponent

3.4.2. Опис EncryptionService та шифрування даних

Однією з важливих можливостей цієї системи для нотування є повне шифрування особистих записів. Це було реалізовано за допомогою Web Cryptography API, функції отримання ключа PBKDF2 та алгоритму шифрування AES-128. Функції отримання ключа, шифрування та дешифрування були об’єднані у класі EncryptionService. Розглянемо його методи:

Публічні методи:

1. async setKey(username, password) – створює ключ шифрування з логіну та паролю та зберігає. Повертає пустий Promise
2. async encrypt(plaintext) – шифрує дану строку тексту. Використовується для текстових даних. Повертає Promise з шифротекстом
3. async encryptBinary(plaintext: ArrayBuffer) - шифрує дані. Використовується для бінарних даних. Повертає Promise з шифротекстом
4. async decrypt(ciphertext: string) – дешифрує дані. Використовується для текстових даних. Повертає Promise з текстом

5. async decryptBinary(ciphertext: string) – дешифрує дані. Використовується для бінарних даних. Повертає Promise з буфером, що містить бінарні дані

Загальний алгоритм шифрування:

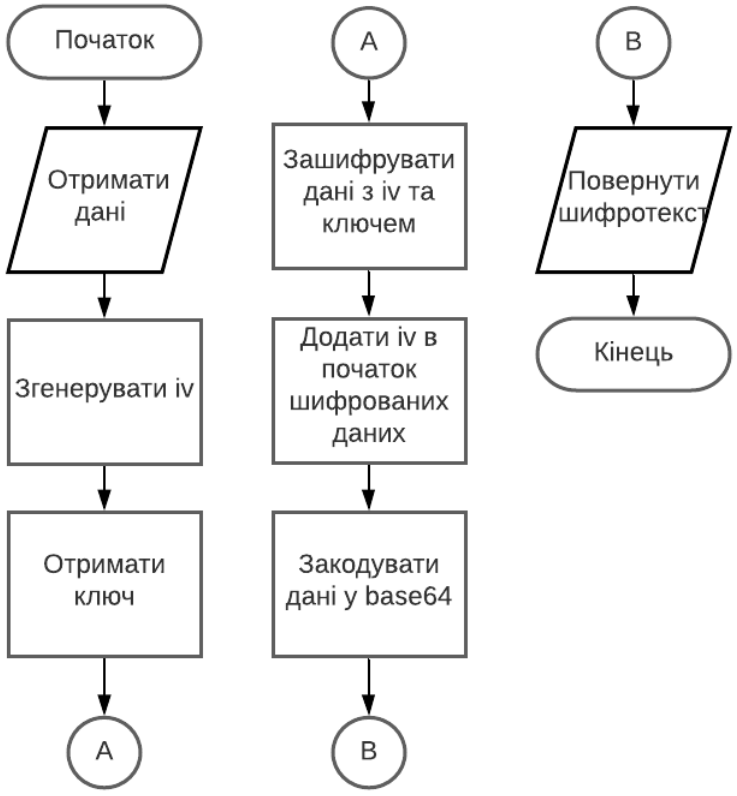


Рис. 3.13 Алгоритм шифрування

Функції encrypt та encryptBinary працюють наступним чином: формується вектор ініціалізації (iv) фіксованої довжини (96 байт) з випадковими значеннями. Отримується збережений ключ. Виконується шифрування алгоритмом AES-GCM, до отриманого масиву даних на початок додається iv. Отримані дані кодуються у base64 для спрощення передачі та зберігання.

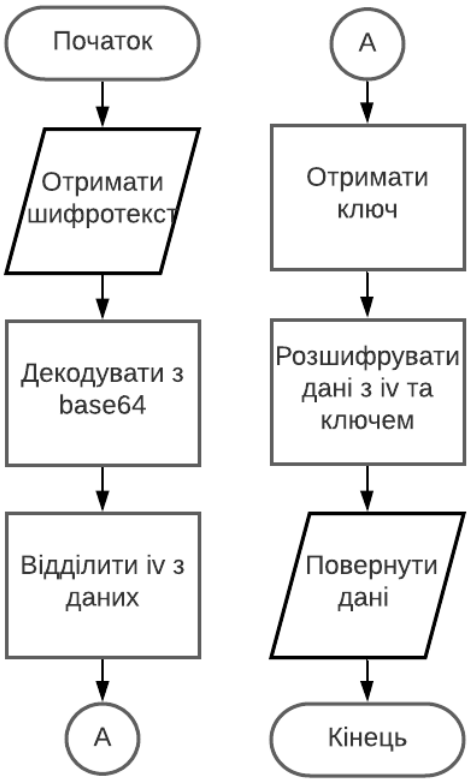


Рис. 3.14 Алгоритм розшифрування

Функції decrypt та decryptBinary працюють навпаки. Спочатку шифротекст декодується з формату base64. Після цього від шифрованих даних відділяється перші 96 байт для отримання iv. Дані дешифруються та повертаються.

3.4.3. Опис перехоплювачів запитів (interceptors)

Так як механізм поновлення та використання access токенів має бути прозорим для користувача та використовується на всіх запитах до сервера окрім /signin, /signup та /refresh, вдалою ідеєю є створення наскрізної логіки. Вона була реалізована за допомогою 2 переоплювачів: applyTokenInterceptor, що додає заголовок Authorization з access токеном до запитів, та RefreshTokenInterceptor, що відповідає за оновлення access токенів та перезапуск запитів, що повертаються з помилкою через закінчення терміну роботи access токена. Розглянемо алгоритм роботи RefreshTokenInterceptor.

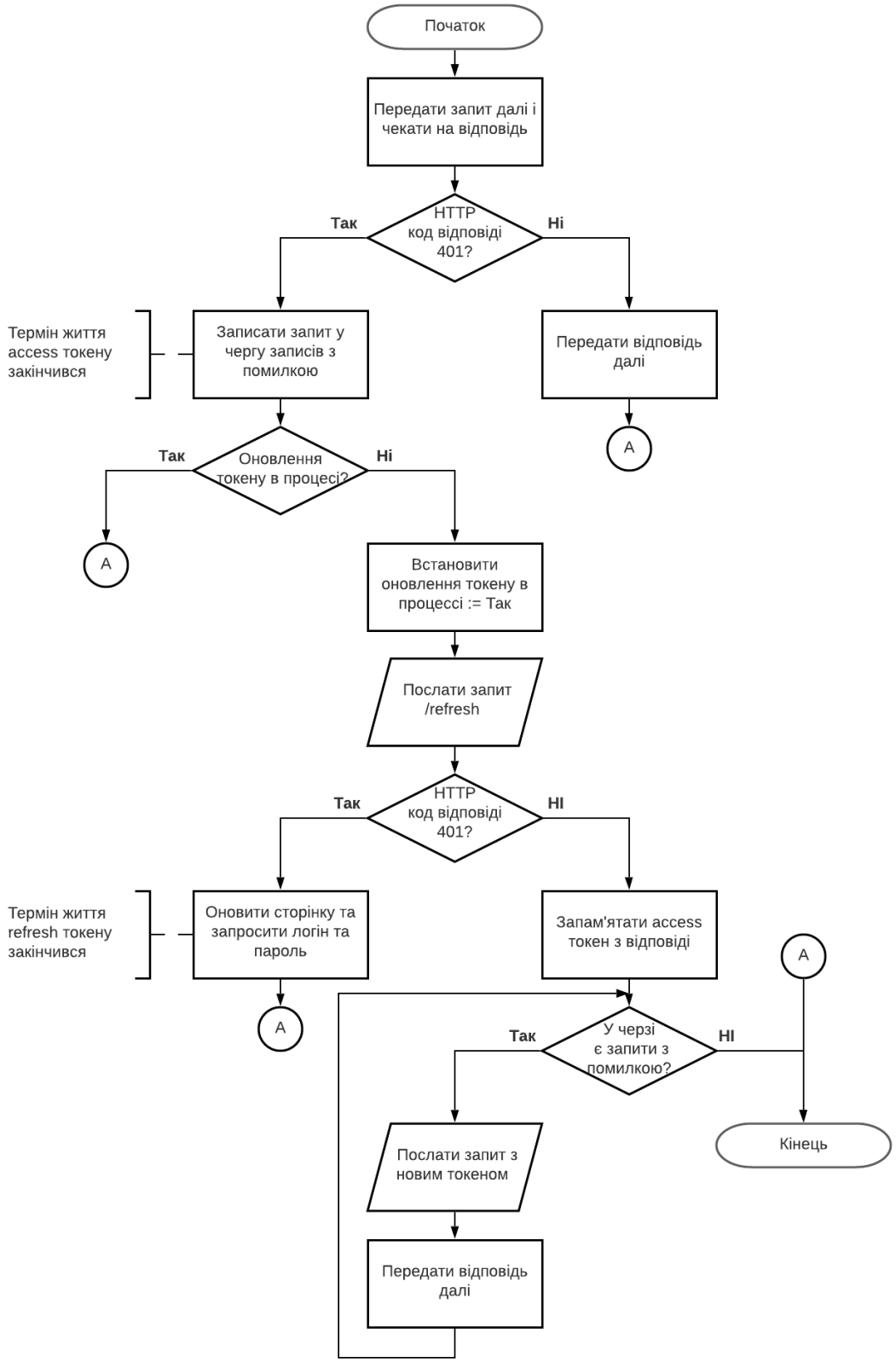


Рис. 3.15 Алгоритм роботи RefreshTokenInterceptor

Цей перехоплювач працює наступним чином: якщо відповідь повертається з будь-яким HTTP кодом, окрім 401, він не робить нічого. Інакше він записує цей запит до черги та перевіряє, чи триває операція поновлення токену. Якщо ні, він починає цю операцію, посилаючи запит на поновлення з refresh токеном. Якщо повертається код 401, то термін життя refresh токену закінчився, і необхідний повторний вхід. Інакше він записує отриманий access токен та перезапускає усі запити, що знаходяться в черзі.

ВИСНОВКИ ДО РОЗДІЛУ 3

У даному розділі був описаний процес розробки клієнт-серверної системи для нотування. Була описані:

1. Доменна модель
2. Загальна архітектура серверу
3. API серверу
4. Поля сутностей доменної моделі, що мають шифруватися для зберігання.
5. Розроблений механізм авторизації та аутентифікації за допомогою access та refresh токенів.
6. Загальна архітектура та компоненти клієнту
7. Розроблений механізм шифрування даних
8. Розроблений механізм використання access та refresh токенів на клієнті

РОЗДІЛ 4
ВИКОРИСТАННЯ СИСТЕМИ

Інструкція користувача є важливою частиною будь-якої програми. У цьому розділі будуть описані можливі сценарії використання створеної системи для нотування

4.1. Вхід до системи

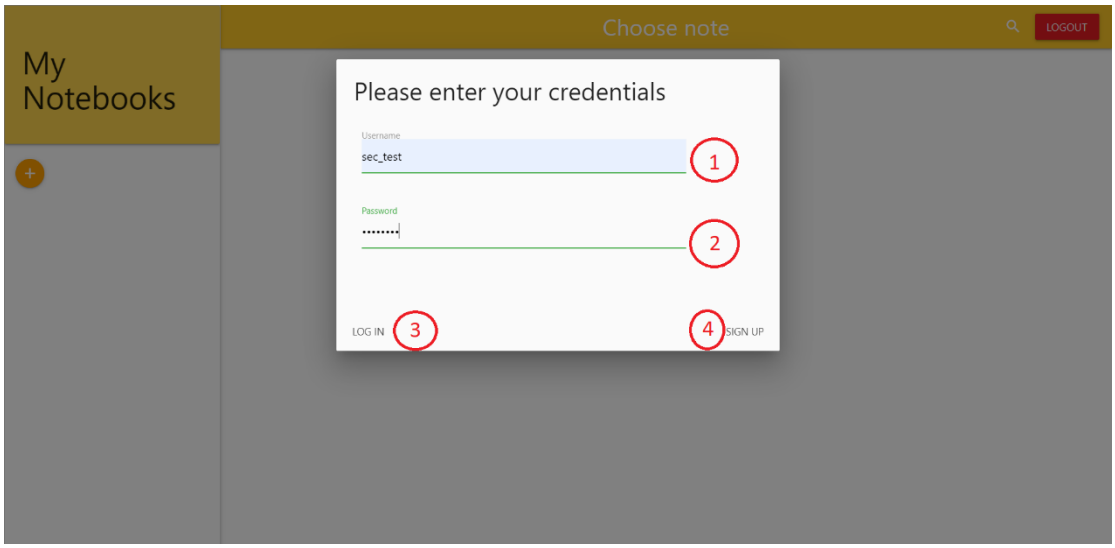


Рис. 4.1 - Вхід до системи

Для того, щоб увійти до системи, перейдіть на сайт системи. Після загрузки ви побачите вікно, в яке ви маєте увести свої логін (1) та пароль (2). Якщо ви вже маєте обліковий запис, натисніть кнопку “LOG IN” (3), інакше - створіть новий, натиснувши “SIGN UP” (4)

4.2. Вибір блокнотів та нотаток

Після входу до системи, користувач має вибрати нотатку. Для цього скористайтеся навігаційним меню зліва. (На мобільних екранах, «витягніть» його жестом зліва).

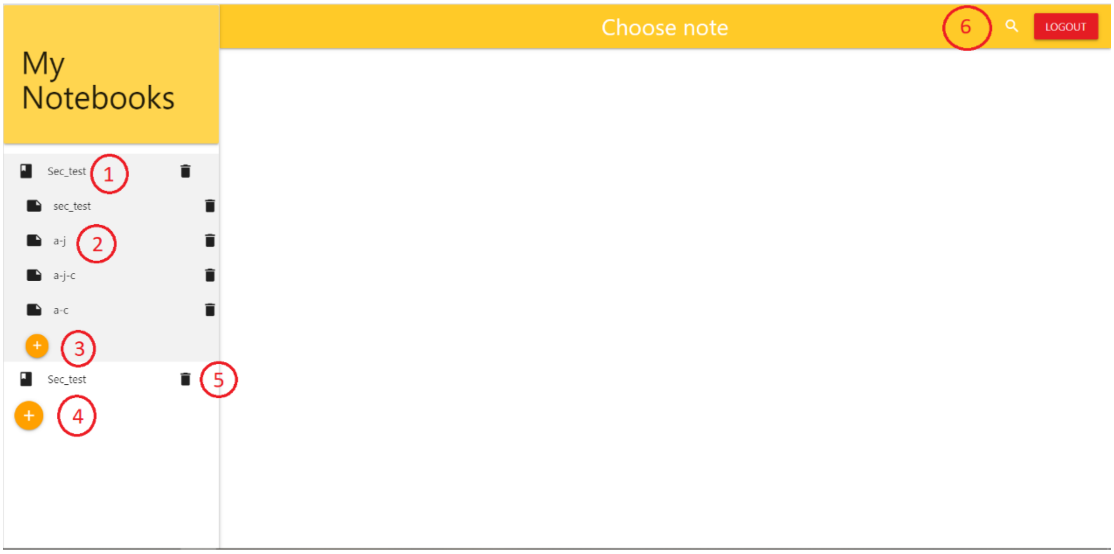


Рис. 4.2 – Навігація

Виберіть блокнот з нотатками, натиснувши на нього (1), після цього виберіть нотатку таким же чином (2). Користувач може додати нові нотатки за допомогою кнопки «+» у кінці списку нотаток (3), та нові блокноти за допомогою кнопки «+» у кінці списку блокнотів (4). Користувач може видалити нотатки та блокноти за допомогою кнопки «видалити» (5). Крім того, користувач може знайти необхідну нотатку за допомогою пошуку по тегам, натиснувши кнопку (6).

4.3. Редагування нотаток

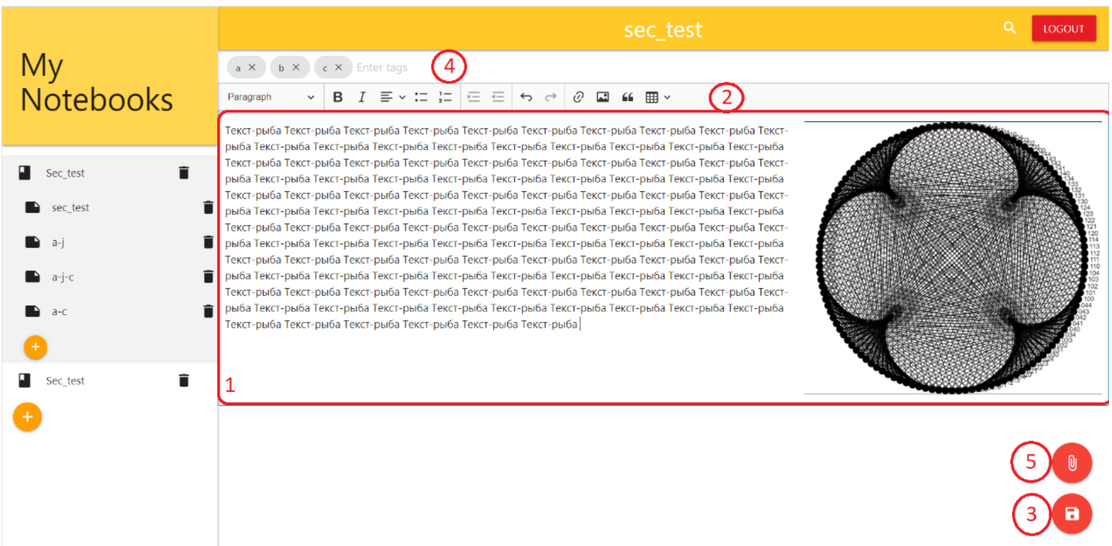


Рис. 4.3 Редагування нотаток

Після вибору нотатки, вона буде відкрита у редакторі, а її назва – відображена у заголовку. Ви можете змінювати її у полі редактору (1). Для цього користувач може скористатися багатьма інструментами редагування (2). Після завершення редагування, збережіть прогрес, натиснувши на кнопку (3). Користувач може додати теги до нотаток (4). Теги чутливі до регістру. Окрім того, є можливість додати файли, натиснувши на кнопку (5)

4.4. Додавання файлів до нотаток

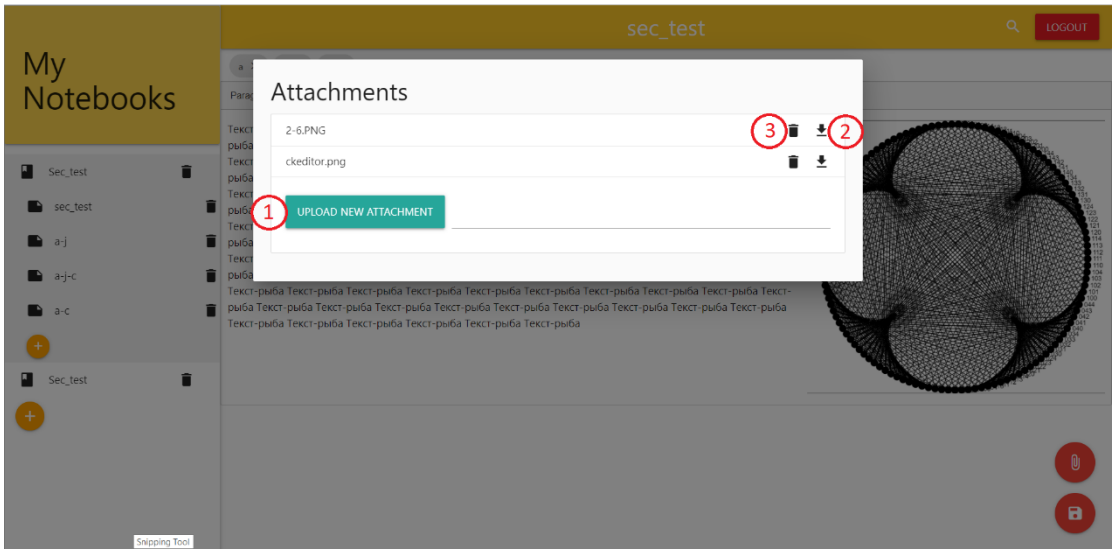


Рис. 4.4 Додавання файлів до нотаток

Додавання файлів можливо у модальному вікні, у якому відображаються всі вкладені у нотатку файли. Для того, щоб додати файл, користувач має натиснути кнопку (1) і вибрати файл. Для того, щоб завантажити файл з сервера, натисніть (2). Для того, щоб видалити файл, натисніть (3)

4.5. Пошук нотаток за тегами

Окрім блокнотів, нотатки також можна знайти за тегами. У відкритому модальному вікні введіть всі шукані теги у стрічку (1). При введенні будуть з’являться підказки – теги, що користувач

використовує в своїх нотатках. Після введення всіх тегів натисніть кнопку “Search” (3). Результати з’являться нижче (4). Натисніть на один з них, щоб перейти до заданої нотатки

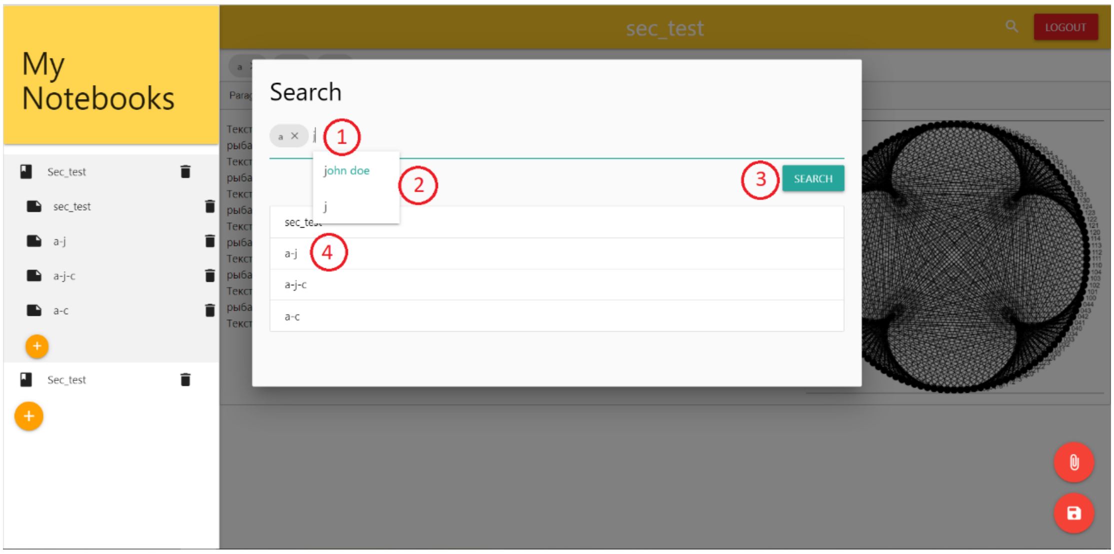


Рис. 4.5 Пошук за тегами

4.6. Вихід з системи

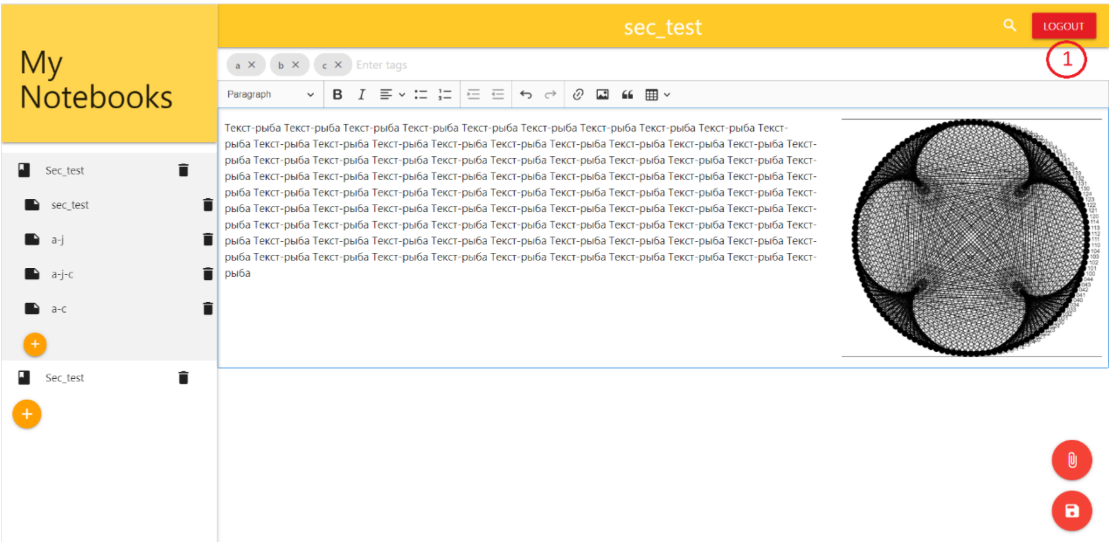


Рис. 4.6 Вихід з системи

Після завершення роботи з нотатками, вийдіть з системи, натиснувши кнопку “LOGOUT” (1)

ВИСНОВКИ ДО РОЗДІЛУ 4

У цьому розділі була розроблена інструкція користувача розробленої системи для нотування. Були описані основні сценарії користування цією програмою.

Інструкція користувача є дуже стислою, але вона надає огляд основних функцій системи. Крім того, завдяки використанню Material Design інтерфейс є більш звичним та інтуїтивним для користувача

ВИСНОВКИ

Метою цієї дипломної роботи є розробка системи для нотування. Після аналізу можливих рішень обрано клієнт-серверна архітектура з монолітним сервером.

У першому розділі проаналізовані різні підходи до організації нотаток, редагування даних та різні підходи до захисту даних. Крім того, розглянуті існуючі програми для нотування, виділені основні їх особливості. Визначені функції, сукупність яких не реалізована в жодній з наявних програм – повне шифрування даних та WYSIWYG редактор, та визначений набір функцій системи, що розробляється.

У другому розділі проаналізовані програмні інструменти для розробки програми для нотування. Визначені модулі, мови програмування та фреймворки, що використані під час розробки системи для нотування, а саме база даних PostgreSQL, фреймворк Spring для розробки серверу та фреймворк Angular для клієнта. Для WYSIWYG-редактору на клієнті обрана спеціальна збірка CKEditor 5.

У третьому розділі описаний процес розробки системи. Наведена схема бази даних (модель даних), описане розроблене API серверу, описаний механізм автентифікації та авторизації за допомогою токенів як зі сторони серверу, так і зі сторони клієнту, описані компоненти клієнту та алгоритми шифрування даних. Наведена принципова схема роботи перехоплювача запитів, що використовується для поновлення access токенів.

У четвертому розділі наведена розроблена інструкція користувача системи для нотування та розглянуті основні сценарії використання.

Розроблена система займає описану нішу на ринку, надаючи захист повного шифрування даних разом зі зручністю WYSIWYG-редактору. Завдяки використанню Material Design, вона має звичний та інтуїтивний інтерфейс користувача.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Rodney H. Jones, Christoph A. Hafner Understanding Digital Literacies - Routledge, 2012р. – с.23
2. Определение WYSIWYG в кембриджском словаре английского языка [Електронний ресурс]. Режим доступу: https://dictionary.cambridge.org/ru/словарь/английский/wysiwyg (дата звертання 03.06.2020)
3. Велика українська енциклопедія: в 30т - К. : Державна наукова установа «Енциклопедичне видавництво», 2016. — Т.1 «А – Акц» -592 с
4. Ferguson N., Schneier B., Kohno T. Cryptography Engineering: Design Principles and Practical Applications - John Wiley & Sons, 2010 – 384 с.
5. OneNote – програма для цифрових нотаток Office [Електронний ресурс]. Режим доступу: https://www.microsoft.com/uk-ua/microsoft-365/onenote/digital-note-taking-app?rtc=1 (дата звертання 03.06.2020)
6. Лучшее приложение для создания заметок – Организуйте заметки с Evernote [Електронний ресурс]. Режим доступу: https://evernote.com/intl/ru/ (дата звертання 03.06.2020)
7. Turtl: The secure, collaborative notebook | Turtl [Електронний ресурс]. Режим доступу: https://turtlapp.com/ (дата звертання 03.06.2020)
8. CKEditor | Smart WYSIWYG HTML editor | Collaborative rich text editor [Електронний ресурс]. Режим доступу: https://ckeditor.com/ (дата звертання 03.06.2020)
9. Web Cryptography API [Електронний ресурс]. Режим доступу:

https://www.w3.org/TR/WebCryptoAPI/ (дата звертання 03.06.2020)

10. Cosmina I., Harrop R., Schaefer C., Ho C. Pro Spring 5: An In-Depth Guide to the Spring Framework and Its Tools – Apress, 2017 – 849 c.
11. PostgreSQL: Documentation: 12: 2. A Brief History of PostgreSQL [Електронний ресурс]. Режим доступу: https://www.postgresql.org/docs/current/history.html (дата звертання 03.06.2020)

ДОДАТОК 1

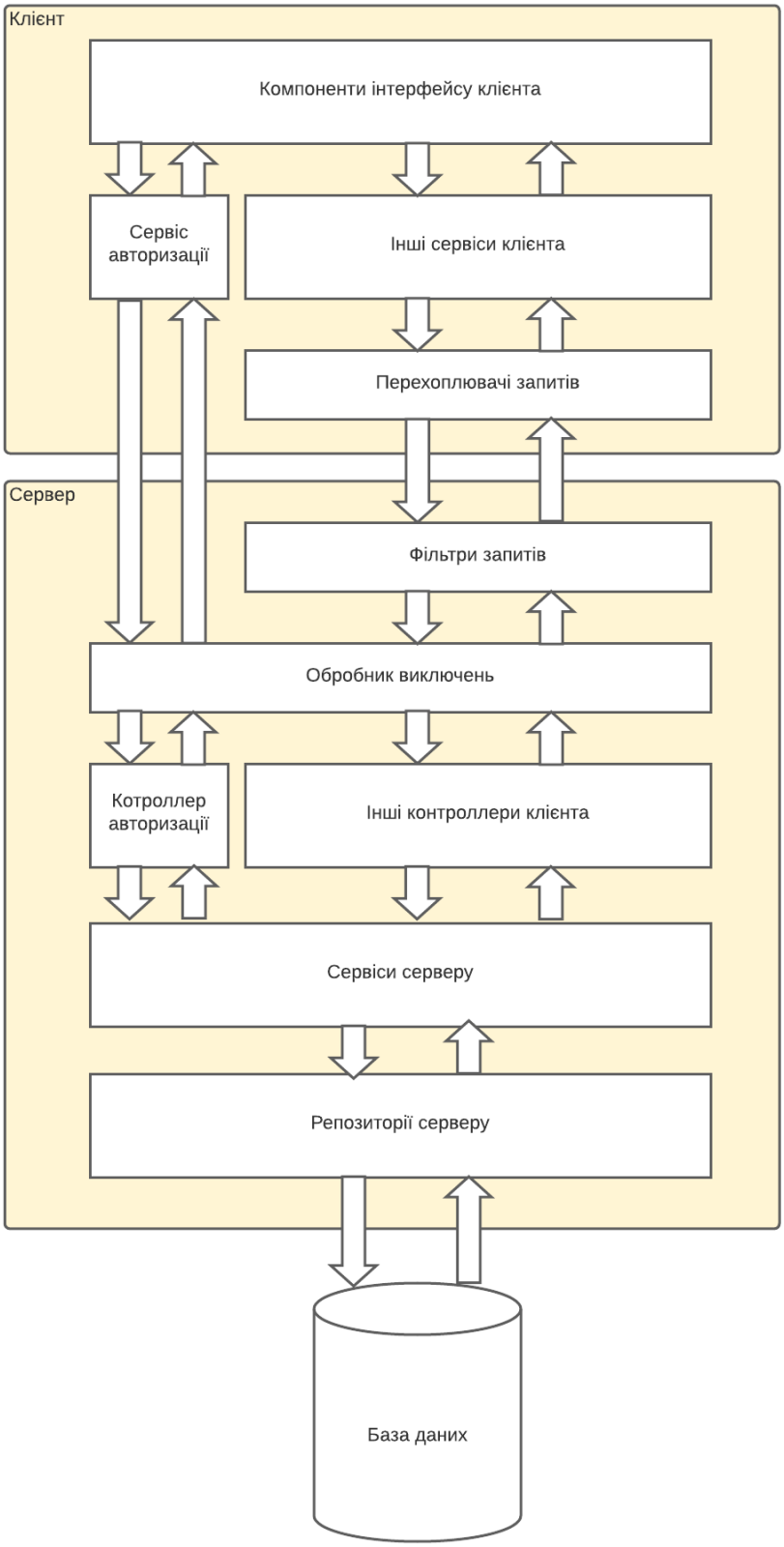
Система управління особистими записами

Схема структурна

ІАЛЦ.467200.005 Д1

Аркушів 1

Київ 2020 р



					ІАЛЦ.467200.003 Д1		
Зм.	Арк.	№ докум.	Підпис	Дата			
Разробив		Кравець П.А.			Система керування особистими записами Схема структурна	Літ.	Аркуш 73 / Аркушів 1
Керівник		Волокита А.М.					
Реценз.						НТУУ “КПІ ім. Сикорського” ФІОТ	
Н. Контр.		Сімоненко В.П.					
Затв.							

ДОДАТОК 2

Система управління особистими записами

Схема функціональна – діаграма класів

ІАЛЦ.467200.004 Д2

Аркушів 1

Київ 2020 р

ІАЛЦ 467200.0042

					ІАЛЦ 467200.004 Д2	
Зм.	Арк.	№ докум.	Підпис	Дата	*Схема структурна Діаграма класів*	Літ. / Маса / Масштаб
Розроб.		Кравець П.А.				
Перевір.		Волокита А.М				Арк. / Аркушів
Н. контр.		Симоненко В.П.			*Дипломна робота*	*НТУУ "КПІ ім. Сікорського" ФІОТ*
Затверд.						

ДОДАТОК З

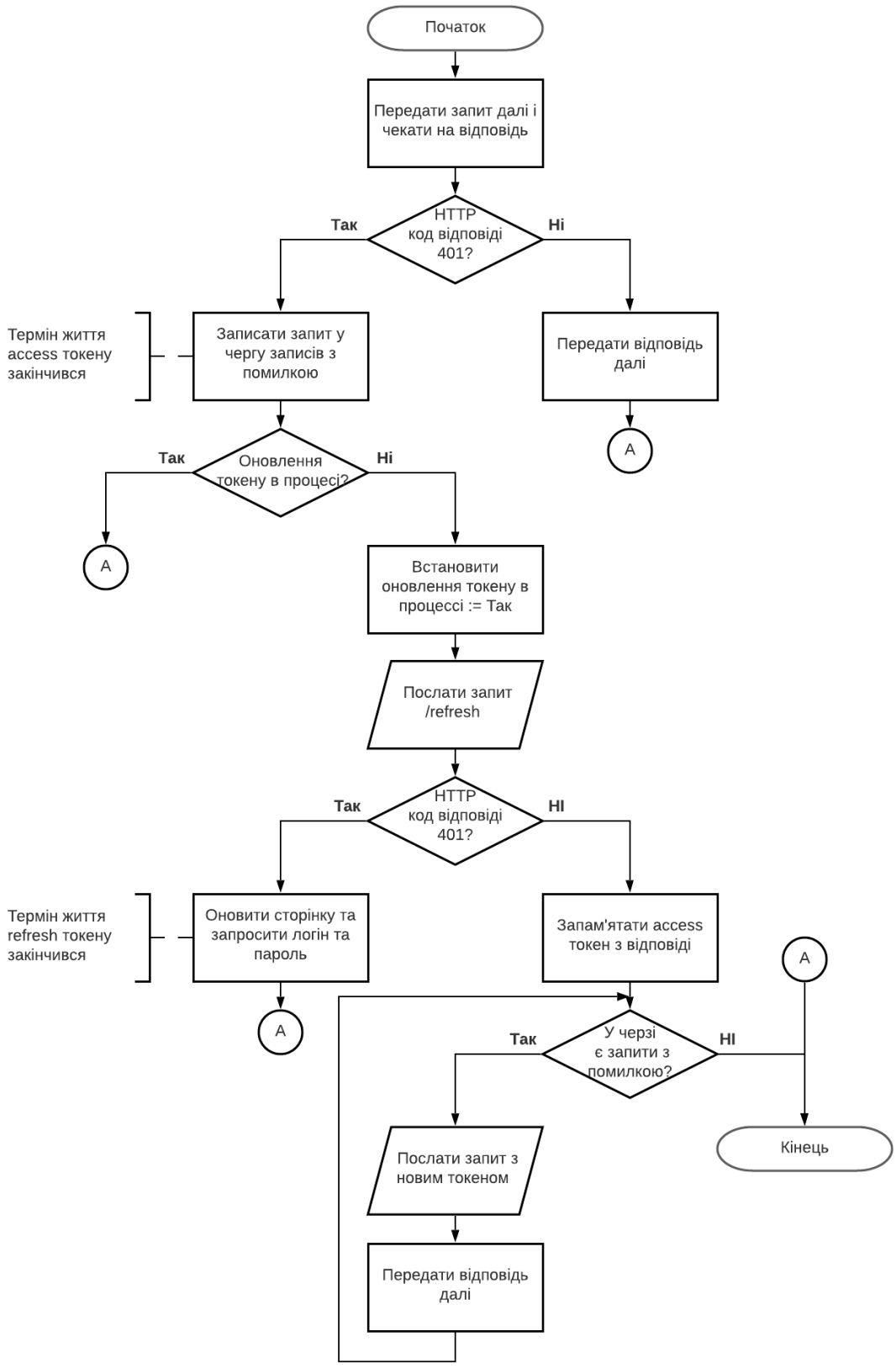
Система управління особистими записами

Схема принципова – схема алгоритму

ІАЛЦ.467200.005 ДЗ

Аркушів 1

Київ 2020 р



					ІАЛЦ.467200.005 ДЗ	
Зм.	*Арк.*	*№ докум.*	*Підпис*	*Дата*		
Разробив		*Кравець П.А.*			Схема принципова Алгоритм перехоплювача запитів	*Літ.* / *Аркуш* / *Аркушів*: 77 / 1
Керівник		*Волокита А.М.*				
Реценз.						*НТУУ "КПІ ім. Сикорського" ФІОТ*
Н. Контр.		*Сімоненко В.П.*				
Затв.						

ДОДАТОК 4

Система управління особистими записами

Лістинг частини коду програми

ІАЛЦ.467200.004 Д4

Аркушів 1

Київ 2020 р

AuthController.java

```
package edu.kpi.notetaker.controller;

import edu.kpi.notetaker.message.AuthInputMessage;
import edu.kpi.notetaker.message.AuthOutputMessage;
import edu.kpi.notetaker.service.AuthService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class AuthController {
    private final AuthService authService;

    @Autowired
    public AuthController(AuthService authService) {
        this.authService = authService;
    }

    @PostMapping("/signin")
    public AuthOutputMessage signIn(@RequestBody AuthInputMessage message){
        return authService.signIn(message.getUsername(), message.getPassword());
    }

    @PostMapping("/signup")
    public AuthOutputMessage signUp(@RequestBody AuthInputMessage message){
        return authService.signUp(message.getUsername(), message.getPassword());
    }

    @PostMapping("/refresh")
    public String refresh(@RequestBody String refreshToken){
        return authService.refresh(refreshToken);
    }

    @PostMapping("/signout")
    @PreAuthorize("hasAuthority('ROLE_USER')")
    public void signout(){
        authService.signout();
    }
}
```

AuthInputMessage.java

```
package edu.kpi.notetaker.message;

import com.fasterxml.jackson.annotation.JsonProperty;
import lombok.Data;

@Data
public class AuthInputMessage {
    @JsonProperty
    private String username;
    @JsonProperty
    private String password;
}
```

AuthOutputMessage.java

```
package edu.kpi.notetaker.message;

import com.fasterxml.jackson.annotation.JsonProperty;
import lombok.Data;

@Data
public class AuthOutputMessage {
    @JsonProperty
    private String accessToken;
    @JsonProperty
    private String refreshToken;
}
```

AuthService.java

```
package edu.kpi.notetaker.service;

import edu.kpi.notetaker.message.AuthOutputMessage;

public interface AuthService {
    AuthOutputMessage signIn(String username, String password);

    AuthOutputMessage signUp(String username, String password);

    String refresh(String refreshToken);

    void signout();
}
```

AuthServiceImpl.java

```
package edu.kpi.notetaker.service.impl;

import edu.kpi.notetaker.exceptionhandling.exceptions.TokenExpiredException;
import edu.kpi.notetaker.message.AuthOutputMessage;
import edu.kpi.notetaker.model.Role;
import edu.kpi.notetaker.model.User;
import edu.kpi.notetaker.security.JWTProvider;
import edu.kpi.notetaker.service.AuthService;
import edu.kpi.notetaker.service.RoleService;
import edu.kpi.notetaker.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.List;

@Service
public class AuthServiceImpl implements AuthService {
    private final AuthenticationManager authenticationManager;
    private final JWTProvider jwtProvider;
    private final PasswordEncoder passwordEncoder;
    private final UserService userService;
    private final RoleService roleService;
```

```
    @Autowired
    public AuthServiceImpl(AuthenticationManager authenticationManager, JWTProvider jwtProvider,
                           PasswordEncoder passwordEncoder, UserService userService, RoleService
roleService) {
        this.authenticationManager = authenticationManager;
        this.jwtProvider = jwtProvider;
        this.passwordEncoder = passwordEncoder;
        this.userService = userService;
        this.roleService = roleService;
    }

    @Override
    public AuthOutputMessage signIn(String username, String password) {
        Authentication authentication = authenticationManager.authenticate(new
UsernamePasswordAuthenticationToken(
                username, password));

        String refreshToken = jwtProvider.createRefreshToken(username);
        userService.updateUserToken(username, refreshToken);
        SecurityContextHolder.getContext()
                .setAuthentication(authentication);
        UserDetails userDetails = (UserDetails) authentication.getPrincipal();
        AuthOutputMessage message = new AuthOutputMessage();
        message.setAccessToken(jwtProvider.createAccessToken(username));
        message.setRefreshToken(refreshToken);
        return message;
    }

    @Override
    public AuthOutputMessage signUp(String username, String password) {
        User user = new User();
        user.setUsername(username);
        user.setPassword(passwordEncoder.encode(password));
        ArrayList<Role> roles = new ArrayList<>();
        roles.add(roleService.findByName("USER"));
        user.setRoles(roles);
        userService.createUser(user);

        return signIn(username, password);
    }

    @Override
    public String refresh(String refreshToken) {
        String claimedUsername = jwtProvider.getSubject(jwtProvider.parseClaims(refreshToken));
        if(!refreshToken.equals(userService.findByUsername(claimedUsername).getRefreshToken()))
            throw new TokenExpiredException("Refresh token expired");
        return jwtProvider.createAccessToken(claimedUsername);
    }

    @Override
    public void signout() {
        User user = (User) SecurityContextHolder.getContext()
                .getAuthentication().getPrincipal();
        userService.updateUserToken(user.getUsername(), null);
    }
}
```

UserService.java

```
package edu.kpi.notetaker.service;

import edu.kpi.notetaker.model.User;
import org.springframework.security.core.userdetails.UserDetailsService;

public interface UserService extends UserDetailsService {
```

```
    User findById(Integer userId);

    User findByUsername(String username);

    User createUser(User user);

    User updateUserToken(String username, String token);
}
```

UserServiceImpl.java

```
package edu.kpi.notetaker.service.impl;

import edu.kpi.notetaker.exceptionhandling.exceptions.EntityAlreadyExistsException;
import edu.kpi.notetaker.exceptionhandling.exceptions.EntityNotFoundException;
import edu.kpi.notetaker.model.User;
import edu.kpi.notetaker.repository.UserRepository;
import edu.kpi.notetaker.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

import java.time.LocalDateTime;

@Service
public class UserServiceImpl implements UserService {

    private final UserRepository userRepository;

    @Autowired
    public UserServiceImpl(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    @Override
    public User findById(Integer userId) {
        return userRepository.findById(userId)
                .orElseThrow(() -> new EntityNotFoundException("User with id=" + userId + " is  not
found"));
    }

    @Override
    public User findByUsername(String username) {
        return userRepository.findByUsername(username)
                .orElseThrow(() -> new EntityNotFoundException("User with username=" + username + "
is not found"));
    }

    @Override
    public User createUser(User user) {
        userRepository.findByUsername(user.getUsername())
                .ifPresent(x -> {
                    throw new EntityAlreadyExistsException("User with username=" +
user.getUsername() + " already exists");
                });
        user.setCreationTimestamp(LocalDateTime.now());
        return userRepository.save(user);
    }

    @Override
    public User updateUserToken(String userName, String token) {
        User user = findByUsername(userName);
        user.setRefreshToken(token);
```

```
        return userRepository.save(user);
    }

    @Override
    public UserDetails loadUserByUsername(String s) throws UsernameNotFoundException {
        try {
            return findByUsername(s);
        } catch (EntityNotFoundException ex){
            throw new UsernameNotFoundException(ex.getMessage(), ex);
        }
    }
}
```

RoleService.java

```
package edu.kpi.notetaker.service;

import edu.kpi.notetaker.model.Role;

public interface RoleService {
    Role findByName(String name);
}
```

RoleServiceImpl.java

```
package edu.kpi.notetaker.service.impl;

import edu.kpi.notetaker.exceptionhandling.exceptions.EntityNotFoundException;
import edu.kpi.notetaker.model.Role;
import edu.kpi.notetaker.repository.RoleRepository;
import edu.kpi.notetaker.service.RoleService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class RoleServiceImpl implements RoleService {
    private final RoleRepository roleRepository;

    @Autowired
    public RoleServiceImpl(RoleRepository roleRepository) {
        this.roleRepository = roleRepository;
    }

    @Override
    public Role findByName(String name) {
        return roleRepository.findByName(name)
                .orElseThrow(() -> new EntityNotFoundException("Role with name=" + name + " is not
found"));
    }
}
```

UserRepository.java

```
package edu.kpi.notetaker.repository;

import edu.kpi.notetaker.model.User;
import org.springframework.data.repository.CrudRepository;

import java.util.Optional;

public interface UserRepository extends CrudRepository<User, Integer> {
    Integer countBy();
```

```
    Optional<User> findByUsername(String username);
}
```

RoleRepository.java

```
package edu.kpi.notetaker.repository;

import edu.kpi.notetaker.model.Role;
import org.springframework.data.repository.CrudRepository;

import java.util.Optional;

public interface RoleRepository extends CrudRepository<Role, Integer> {
    Optional<Role> findByName(String name);
}
```

User.java

```
package edu.kpi.notetaker.model;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;

import javax.persistence.*;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.Collection;
import java.util.stream.Collectors;

@Data
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "users")
public class User implements UserDetails {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    @Column(unique = true, nullable = false)
    private String username;
    @Column(nullable = false)
    private String password;
    private String refreshToken;
    @Column(name = "creation_timestamp")
    private LocalDateTime creationTimestamp;

    @OneToMany(mappedBy = "user", fetch = FetchType.LAZY, cascade = CascadeType.ALL)
    private Collection<Notebook> notebooks = new ArrayList<>();

    @ManyToMany(fetch = FetchType.EAGER, cascade = CascadeType.MERGE)
    @JoinTable(
            name = "users_roles",
            joinColumns = @JoinColumn(name = "user_id"),
            inverseJoinColumns = @JoinColumn(name = "role_id"))
    private Collection<Role> roles = new ArrayList<>();

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return roles.stream()
                .map(x -> new SimpleGrantedAuthority("ROLE_"+x.getName()))
```

```
                .collect(Collectors.toList());
    }

    @Override
    public boolean isAccountNonExpired() {
        return true;
    }

    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return true;
    }
}
```

Role.java

```
package edu.kpi.notetaker.model;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import javax.persistence.*;

@Data
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "roles")
public class Role {
    @Id
    private Integer id;
    @Column(unique = true, nullable = false)
    private String name;
}
```

SecurityConfig.java

```
package edu.kpi.notetaker.security;

import edu.kpi.notetaker.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpStatus;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.AuthenticationProvider;
import org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import
org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilde
r;
import
org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;
```

```
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
import
org.springframework.security.web.authentication.logout.HttpStatusReturningLogoutSuccessHandler;
import org.springframework.web.cors.CorsConfiguration;
import org.springframework.web.cors.CorsConfigurationSource;
import org.springframework.web.cors.UrlBasedCorsConfigurationSource;

import java.util.Arrays;
import java.util.Collections;

@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    private final UserService userService;
    private final JWTFilter jwtFilter;

    @Autowired
    public SecurityConfig(UserService userService, JWTFilter jwtFilter) {
        this.userService = userService;
        this.jwtFilter = jwtFilter;
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.csrf()
                .disable()
                .cors()
                .and()
                .authorizeRequests()
                .anyRequest()
                .permitAll()
                .and()
                .sessionManagement()
                .sessionCreationPolicy(SessionCreationPolicy.STATELESS);
        http.addFilterBefore(jwtFilter, UsernamePasswordAuthenticationFilter.class);
    }

    @Bean
    CorsConfigurationSource corsConfigurationSource() {
        CorsConfiguration configuration = new CorsConfiguration();
        configuration.setAllowedOrigins(Collections.singletonList("*"));
        configuration.setAllowedMethods(Arrays.asList("GET", "POST", "OPTIONS", "DELETE", "PUT",
"PATCH"));
        configuration.setAllowedHeaders(Arrays.asList("X-Requested-With",
                "Origin",
                "Content-Type",
                "Accept",
                "Authorization",
                "Authorization-Refresh",
                "New-Access-Token",
                "X-Total-Count"));
        configuration.setAllowCredentials(false);
        configuration.addExposedHeader("New-Access-Token");
        configuration.addExposedHeader("X-Total-Count");
        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration("/**", configuration);
        return source;
    }
```

```
    @Override
    protected void configure(AuthenticationManagerBuilder auth) {
        auth.authenticationProvider(authenticationProvider());
    }

    @Bean
    public AuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider auth = new DaoAuthenticationProvider();
        auth.setUserDetailsService(userService);
        auth.setPasswordEncoder(passwordEncoder());
        return auth;
    }

    @Bean
    @Override
    public AuthenticationManager authenticationManagerBean() throws Exception {
        return super.authenticationManagerBean();
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```

JWTProvider.java

```
package edu.kpi.notetaker.security;

import edu.kpi.notetaker.exceptionhandling.exceptions.TokenExpiredException;
import edu.kpi.notetaker.model.User;
import edu.kpi.notetaker.service.UserService;
import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jws;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.PropertySource;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.stereotype.Component;

import java.util.Date;

@Component
@PropertySource(value = {"classpath:jwt.properties"})
public class JWTProvider {
    @Value("${jwt.secret}")
    private String jwtSecret;
    @Value("${jwt.access.validity}")
    private long accessValidityInMilliseconds;
    @Value("${jwt.refresh.validity}")
    private long refreshValidityInMilliseconds;

    private final UserService userService;

    @Autowired
    public JWTProvider(UserService userService) {
        this.userService = userService;
    }

    public Authentication getAuthentication(Jws<Claims> claims){
        User user = userService.findByUsername(getSubject(claims));
```

```
        return new UsernamePasswordAuthenticationToken(user, "", user.getAuthorities());
    }

    public String createAccessToken(String username){
        Claims claims = Jwts.claims().setSubject(username);

        Date now = new Date();
        Date validity = new Date(now.getTime() + accessValidityInMilliseconds);

        return Jwts.builder()
                .setClaims(claims)
                .setIssuedAt(now)
                .setExpiration(validity)
                .signWith(SignatureAlgorithm.HS256, jwtSecret)
                .compact();
    }

    public String createRefreshToken(String username){
        Claims claims = Jwts.claims().setSubject(username);

        Date now = new Date();
        Date validity = new Date(now.getTime() + refreshValidityInMilliseconds);

        return Jwts.builder()
                .setClaims(claims)
                .setIssuedAt(now)
                .setExpiration(validity)
                .signWith(SignatureAlgorithm.HS256, jwtSecret)
                .compact();
    }

    public String getSubject(Jws<Claims> claims){
        return claims.getBody().getSubject();
    }

    public Jws<Claims> parseClaims(String token) {
        return Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(token);
    }
}
```

JWTFilter.java

```
package edu.kpi.notetaker.security;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.SerializationFeature;
import edu.kpi.notetaker.exceptionhandling.ErrorMessage;
import edu.kpi.notetaker.exceptionhandling.ExceptionHandlerAdvice;
import edu.kpi.notetaker.exceptionhandling.exceptions.TokenExpiredException;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Component;
import org.springframework.web.context.request.ServletWebRequest;
import org.springframework.web.filter.OncePerRequestFilter;

import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;
import java.text.SimpleDateFormat;
import java.time.LocalDateTime;
```

```
@Component
public class JWTFilter extends OncePerRequestFilter {
    private final JWTProvider provider;
    private final ExceptionHandlerAdvice advice;

    @Autowired
    public JWTFilter(JWTProvider provider, ExceptionHandlerAdvice advice) {
        this.provider = provider;
        this.advice = advice;
    }

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain) throws ServletException, IOException {
        String token = resolveToken(request);
        if (token != null) {
            try{
                Authentication auth = provider.getAuthentication(provider.parseClaims(token));
                SecurityContextHolder.getContext().setAuthentication(auth);
            } catch (Exception ex){
                ErrorMessage message = new ErrorMessage(LocalDateTime.now(),
HttpStatus.UNAUTHORIZED,
                        "Access token expired", request.getRequestURI());

                response.setStatus(message.getStatus());
                response.setContentType("application/json");

                ObjectMapper mapper = new ObjectMapper();
                PrintWriter out = response.getWriter();
                out.print(mapper.writeValueAsString(message));
                out.flush();
                return;
            }
        }
        filterChain.doFilter(request, response);
    }

    @Override
    protected boolean shouldNotFilter(HttpServletRequest request) throws ServletException {
        String path = request.getRequestURI();
        return path.equals("/signin") || path.equals("/signup") || path.equals("/refresh");
    }

    private String resolveToken(HttpServletRequest req) {
        return req.getHeader("Authorization");
    }
}
```

ExceptionHandlerAdvice.java

```
package edu.kpi.notetaker.exceptionhandling;

import edu.kpi.notetaker.exceptionhandling.exceptions.EntityAlreadyExistsException;
import edu.kpi.notetaker.exceptionhandling.exceptions.EntityNotFoundException;
import edu.kpi.notetaker.exceptionhandling.exceptions.TokenExpiredException;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.BadCredentialsException;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.context.request.ServletWebRequest;
import org.springframework.web.context.request.WebRequest;
```

```
import org.springframework.web.servlet.mvc.method.annotation.ResponseEntityExceptionHandler;

import java.time.LocalDateTime;

@ControllerAdvice
public class ExceptionHandlerAdvice extends ResponseEntityExceptionHandler {

    @ExceptionHandler(value = {EntityAlreadyExistsException.class})
    protected ResponseEntity<Object> handleConflict(RuntimeException ex, WebRequest request) {
        return handleExceptionInternal(ex,
                new ErrorMessage(LocalDateTime.now(), HttpStatus.CONFLICT, ex.getMessage(),
                        ((ServletWebRequest)request).getRequest().getRequestURI()),
                new HttpHeaders(), HttpStatus.CONFLICT, request);
    }

    @ExceptionHandler(value = {EntityNotFoundException.class})
    protected ResponseEntity<Object> handleNotFound(RuntimeException ex, WebRequest request) {
        return handleExceptionInternal(ex,
                new ErrorMessage(LocalDateTime.now(), HttpStatus.NOT_FOUND, ex.getMessage(),
                        ((ServletWebRequest)request).getRequest().getRequestURI()),
                new HttpHeaders(), HttpStatus.NOT_FOUND, request);
    }

    @ExceptionHandler(value = {TokenExpiredException.class, BadCredentialsException.class})
    protected ResponseEntity<Object> handleUnauthorised(RuntimeException ex, WebRequest request) {
        return handleExceptionInternal(ex,
                new ErrorMessage(LocalDateTime.now(), HttpStatus.UNAUTHORIZED, ex.getMessage(),
                        ((ServletWebRequest)request).getRequest().getRequestURI()),
                new HttpHeaders(), HttpStatus.UNAUTHORIZED, request);
    }
}
```